Efficient Methods for Decentralized Optimization over Graphs

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Meng Ma

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Georgios B. Ginnakis

April, 2021

Acknowledgements

There are so many people to whom I would like to express my gratitude for makeing my years at the University of Minnesota the most rewarding journey.

My deepest gratitude goes to my advisor Prof. Georgios B. Giannakis. His encouragement and support at the beginning of my application for graduate schools embarked me on this amazing journey. In the subsequent years, his insightful advice on research topics and valuable feedback on academic writing and presentation skills guided me through the Ph.D. journey, and his work ethic taught me what an extraordinary researcher should be. It was his continuous support and encouragement that helped me navigate through difficulties and darkness. Without his help, this thesis would not have been possible.

I would like also to thank my committee members Prof. Mingyi Hong, Prof. Andrew Lamperski, and Prof. Yousef Saad for serving on my doctoral committee. I am very grateful for their patience in coordinating the date and their time to attend the defense. Their insightful comments and valuable feedback greatly helped improve the quality of my research.

Throughout the my graduate studies, I have been lucky enough to work with many talented individuals. I benefited a lot from working and discussing with them. In particular, I would like to thank Prof. Daniel Romero (now at University of Agder, Norway), who patently trained me into my research topics during my first year at the University of Minnesota. I would like also to extend my gratitude to my collaborators Dr. Athanasios N. Nikolakopoulos, Dr. Jineng Ren, Prof. Jarvis Haupt, Bingcong Li, Alireza Sadeghi, and Dr. Vassilis Ioannidis. I also want to thank Dr. Yunlong Wang for offering me the opportunity of internship in his team. This thesis has also benefited from discussing with current and former group members of SPiNCOM and visitors: Prof. Tianyi Chen, Prof. Yanning Shen, Dr. Dimitris Berberidis, Dr. Liang Zhang, Prof. Gang Wang, Dr. Donghoon Lee, Dr. Fatemeh Sheikholeslami, Dr. Brian Baingana, Seth Barrash, Prof. Jia Chen, Prof. Yu Zhang, Dr. Panagiotis Traganitis, Georgios Vasileios Karanikolas,

Qiuling Yang, Prof. Antonio G. Marques, Prof. Qing Ling, Prof. Geert Leus, Prof. Gonzalo Mateos, Dr. Mario Coutiño, Prof. Konstantinos Slavakis.

I would like also to express my thanks to my friends that I met in the University of Minnesota, some of whom have already been mentioned in the above: Dr. Bo Yang, Dr. Xinguo Li, Haoji Hu, Zeren Shui, Prof. Zehua Guo, Wen Zhou, Zhengyang Zhao, Xuzhe Ying, Dr. Zhan Lu. They made this journey full of fun and happiness.

Last but not least, I would like to thank my family for their unconditional love, companion, and caring. Particularly, I am thankful to my parents who worked so hard for many years to support my education and always encouraged me to pursue my dream. I would never make this without their love and support.

Meng Ma, Minneapolis, April 2021

Dedication

This dissertation is dedicated to my family and friends for their unconditional love and support.

Abstract

The quick evolution and widespread applicability of machine learning and artificial intelligence have fundamentally shaped and transcended modern life. Three key players stand behind such a ubiquitous emergence: big data, growing computing power, and improved algorithms. The need for distributed storage and processing arises from this "data deluge" that can flood any powerful machine, from the dispersively available data that are prohibitively costly to transfer to a central unit for further processing, and from the prevalent Internet-of-Things (IoT) devices that require real-time response as well as respect of privacy. Modern machine learning algorithms built to exploit such huge amounts of data are often computationally "hungry" and their "appetite" for computing power increases rapidly at a pace unmatched by the development of computing hardware. All these considerations justify the pressing need for distributed optimization algorithms that are scalable yet flexible to adapt to various configurations of networked computing nodes.

To cope with these challenges, the present thesis first introduces a novel ADMM based approach (termed hybrid ADMM) for efficient decentralized optimization. By modeling the underlying communication patterns as hypergraphs, it provides a unifying framework that subsumes both centralized and fully decentralized counterparts as special cases, and allows nodes to communicate in centralized and decentralized ways at the same time. Leveraging the expressiveness of hypergraph models, a technique termed "in-network acceleration" is introduced enabling "almost free" performance gain by exploiting local graph topology. To account for heterogeneity of nodes and edges, a diagonal scaling based approach is proposed to tackle weighted updates, where proper edge weights are identified through solving a preconditioning problem. By assigning larger weights to critical edges, the proposed algorithm achieves higher efficiency and becomes more robust to perturbations. Finally, to boost the efficiency of the whole system to its full potential, an asynchronous method is introduced to mitigate the straggler problem so that nodes with different processing power can run at full speed. Convergence analysis for proposed algorithms is provided which reveals the connection between convergence rate and spectral properties of the communication graph. Numerical tests of several common tasks on different graphs are carried out to demonstrate the effectiveness of proposed algorithms.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation and context

Many contemporary machine learning and signal processing tasks that have to deal with big data can be formulated as distributed optimization problems over networks. Such problems entail parallel processing of data acquired by interconnected nodes such as ubiquitous Internet-of-Things (IoT) devices, see Figure 1.1. Applications include data fusion and processing using sensor networks [116, 90, 66, 100], vehicle coordination [98, 97], power state estimation [57], and regression [77], to name a few. Among the candidate solvers, the *alternating direction method of multipliers* (ADMM) [5, 9] stands out as an efficient and versatile choice that has attracted much research interest in recent years [24, 43, 51, 19], thanks to its simplicity, decomposability, and mild convergence conditions.

Many distributed optimization problems can be formulated in a consensus form and solved efficiently by ADMM [9, 37]. The most popular approach termed *centralized* ADMM (CADMM) requires a global central processing unit coordinating all the nodes [9]. Decentralized optimization on the other hand, Another approach termed *decentralized* ADMM (DADMM) forgoes the central processing unit by exchanging information only between single-hop neighbors, see [37] and reference therein. However, several critical issues must be tackled before embracing the full potential of distributed optimization over large-scale networks. We highlight the following four main challenges of interest to this research.

Figure 1.1: Ubiquitousness of IoT devices.

C1. Network scalability. Although successfully applied in various settings, neither CADMM nor DADMM is designed to cope with large-scale networks. On the one hand, as the network size grows, the cost of connecting every node to the central coordinator becomes prohibitive due to overwhelming communication overhead [62, 112], and the storage requirement could easily surpass the capacity of a single machine. Furthermore, the mere presence of one dedicated central coordinator renders the whole system vulnerable to single-point failures. DADMM, on the other hand, forgoes the central coordinator and thus bypasses these challenges. But for networks with large diameters however, DADMM convergence slows down significantly as reaching remote destinations through multiple neighbor-to-neighbor communications incurs large delays.

C2. Adaptivity to network topology. Both CADMM and DAMM are not able to adapt to the network topology. Specifically, CADMM: (a) applies only to a star topology with one central unit connected to all other nodes; and (b) the central unit does not 'own' its data, but only coordinates computation across nodes. DADMM overcomes these difficulties by only requiring communication among single-hop neighbors and does not rely on the presence of dedicated coordinators [100, 37]. But it goes too far to another extreme that only one-hop neighbors are "visible" to each node. It is this "short-sighted" focus that renders it blind to the "bigger picture," thus hindering its ability to leverage the underlying network topology.

C3. Convergence in practical settings. ADMM was first used for distributed optimization in centralized form [5], and its convergence has been studied extensively [9, 19, 43, 50]. Due the fast growing demand of computing and communication power from ever increasing volume of IoT devices and various system concerns such as congestion, security, and privacy, decentralized algorithms have received more attention, and ADMM in particular has attracted considerable research interests. During the past decades, linear convergence of DADMM has also been established [103, 74, 54, 71]. But the convergence rate results so far are not tight, while the limiting behavior of iterative updates in practice is far from theoretical predictions, making efforts to optimize convergence rate less effective.

C4. Synchronization across network nodes. Both CADMM and DADMM are synchronous algorithms. All nodes need to carry out update and communication steps at exactly the same time, which can be impractical. An extra synchronization algorithm is needed per iteration to maintain synchronization. In fact, the synchronization itself can be even more complex and costly than the problem to be solved. Another challenge with synchronous algorithms is that they are generally inadequate in dealing with the heterogeneity of nodes. The difference in node speed prevents the system running at full speed as faster nodes have to wait for the slower ones.

The key contributions of the present dissertation are algorithms, theories, and relevant techniques that aim to cope with the challenges C1–C4.

1.1.1 Hybrid ADMM for Decentralized Optimization

Decentralized optimization arises frequently in a variety of engineering problems, where networked machines work collectively to minimize some objective while maintaining a common decision variable. Collaboration are only allowed among direct neighbors connected by network edges. Prominent examples includes multi-agent coordination, large scale machine learning, distributed tracking and localization, detection and estimation over sensor networks [82, 83, 100, 98, 9, 37, 89, 13, 58, 78], to name a few. In a nutshell, a decentralized optimization problem aims to minimize some *separable* objective function $f(\boldsymbol{x}) = \sum_{i=1}^{n} f_i(\boldsymbol{x})$ by finding a *common* decision variable $\boldsymbol{x} \in \mathbb{R}^l$ using n networked computing machine. The local objective f_i typically resides on one machine with possibly *private* data not to be shared with others. The *distributed* nature and *coupled* decision variable make it difficult for parallel processing. A common strategy is to create local copies at each node and enforce equality

among all local variables. The decoupled problem allows parallel minimization of each local objective while periodically exchanging information among computing nodes to ensure all local variables reach consensus eventually.

ADMM is well suited for handling such constrained consensus problems, which leads to periodic exchange of information between all pair of connected nodes. Naively applying ADMM, however, yields sub-optimal performance since every node looks at each of its neighbors individually without realizing overall topology of its neighborhood. Moreover, there is no way to achieve something between centralized and decentralized methods, i.e., a mixture of both where some nodes collaborate in centralized way while others only talk to neighbors. To cope with all these challenges, a novel method termed hybrid ADMM is proposed.

Related work. Distributed optimization problem has been extensively studied during the past decades and many algorithms have been proposed since then. One particularly popular class is (sub)gradient related methods, including distributed (sub)gradient descent [82, 99, 100], incremental subgradient [93], projected subgradient [105, 83], dual averaging [21], gossip [20]. Typically, a node updates its local decision variable by incorporating information from its neighbors and then taking a (sub)gradient step based on its local objective. These methods can be quite slow to converge. Furthermore, additional assumptions are necessary to ensure convergence, for example, the weight matrix must be doubly stochastic [82, 105, 83].

The decomposability of ADMM makes it particularly suited for decentralized optimization, and many variants has been developed including centralized [5, 9], decentralized [37, 103], weighted [65] and Nesterov accelerated [43]. Since introduced in the 1970s [42], convergence analysis of ADMM can be found in e.g. [32, 23, 106, 9]. Local linear convergence of ADMM for linear or quadratic programs is established in [6]; see also [51] where the cost is a sum of component costs. Global linear convergence of a more general form of ADMM is reported in [19]; and linear convergence for a generalized formulation of consensus ADMM using the so-called "communication matrix" in [74]. Though ADMM has been applied to various decentralized optimization problems [66, 121, 100], its convergence remained open until linear convergence of ADMM was established in [103] and [65] for its weighted counterpart. A successive orthogonal projection approach for distributed learning over networked nodes is introduced in [92], where nodes cannot communicate, but each node can access only limited amounts of data and agreement is enforced across nodes sharing the same data. A distributed ADMM algorithm

that deals with node clusters was proposed, for which linear convergence was also substantiated [54]. However, it relies on the gossip algorithm for communication within clusters when cluster head is absent, and the explicit rate bound is difficult to obtain; and, it admits no closed-form representation for general networks.

1.1.2 Weighted ADMM for Fast Decentralized Optimization

A graph can be completely specified by its nodes and edges. The hybrid ADMM improves the performance of DADMM by taking into consideration node importance, but it treats all edges equally and thus fails to account for edges importance, which could lead to degraded performance. For example, in a graph of two clusters connected by a single edge, this specific edge becomes critical because all information about the other cluster can only be obtained via this path. As a results, more emphasis should be put on information obtained through this critical edge. Therefore, blindly assuming equal weights for all edges is far from optimal, and some systematic ways to take edge weights into account are yet to be developed.

Related work. Convergence of general ADMM has been studied for a long time and is well understood [9]. However, convergence results for decentralized ADMM remained open until recently due to the linear constraints [51, 19, 103, 74]. The relation between convergence rate of ADMM and the topology of underlying graph is explored using lifted Markov chain [31]. Convergence of hybrid ADMM has also been shown [71]. In the setting of distributed averaging, the optimal weights and step size are considered in [35]. Decentralized ADMM over weighted graphs was also considered [65], which offers the flexibility to optimize convergence rate by tuning weights. The importance of edges was incorporated into hybrid ADMM as well [69] and its convergence rate bound was shown to be related to spectral properties of the communication graph. However, none of these convergence rate results is tight. One cannot find a specific example that the actual convergence behavior matches the theoretical bound.

The problem of finding optimal edge weights has been investigated in various settings. The optimal design of edge weights for conventional decentralized ADMM has been dealt with by optimizing the corresponding convergence rate [65]. For general constrained optimization where $\boldsymbol{A}$ has full row rank, the optimal scaling matrix can be found by minimizing the (cross) condition number of two positive (semi)definite matrices [38, 40], but this cannot be applied to decentralized optimization where $\boldsymbol{A}$ does not have full row rank. Based on the convergence rate

that is related to graph topology, the adjacency matrix of the weighted graph is directly solved by maximizing the graph connectivity [69]. This method only works for certain types of graphs due to the loose convergence rate bound.

1.1.3 Asynchronous Decentralized Optimization with Multiple Masters

Decentralized optimization benefits from local aggregation that exploits spatial information, namely the relative importance of nodes and edges. But it does not consider harnessing temporal local aggregation to further improve performance. So far, all algorithms discussed are synchronous, in the sense that all nodes send out their updates and receive updates at the same time. Due to the heterogeneity in computing power, different machines complete their tasks at a different pace. The synchronization locks the faster machines, keeps them idle while waiting for slower ones, and slows down the speed of the whole system. This motivates well algorithms performing asynchronous local updates to reduce the overall system delay.

Related work. Asynchronous algorithms for distributed optimization using ADMM are also either *centralized* or *decentralized*. Centralized methods assume a communication graph of star topology, which consists of a center node referred as *master* and others as *workers*. Centralized algorithms are popular because they are easier to analyze and implement [120, 16, 49, 91], but the single master operation faces single-point failure and bottleneck related challenges that limit the overall system performance, the same as their synchronous counterparts. Consequently, decentralized alternatives have been developed [114, 91]. But due to the difficulty in analyzing the algorithms in face of complicated asynchronous updates, not too much results about the convergence are available. Moreover, no method is available to accommodate multiple masters except for [53], which is asynchronous version of the cluster-based method [54].

1.2 Thesis outline

The remainder of the thesis is organized as follows.

Chapter 2 puts forth a hybrid ADMM [73, 71] method for decentralized optimization leveraging local aggregation and network topology. The idea is based on the observation that centralized methods [9] tends to converge faster than decentralized ones [66, 28, 27] due to short

path of information flow. The hybrid ADMM provides a unifying framework that generalizes both centralized and decentralized ADMM using a communication graph that encodes the update rules for each node. Such communications graphs subsume both centralized and decentralized methods as special cases, and enables a novel hybrid decentralized optimization approach that offers much more flexibility by allowing a mixture of locally centralized updates and fully decentralized ones, thus gains considerable performance boost. This chapter continues by introducing a novel technique call "in network acceleration" that enables implementation of HADMM over the same physical network as fully decentralized method, offering "almost free" performance improvement without physically deploying any local coordinator. Experiments on different kinds of graphs demonstrate the effectiveness of hybrid ADMM algorithm.

Chapter 3 builds on the hybrid ADMM algorithm and extends its capability to realm of weighted updates [69]. In decentralized algorithms, nodes can only communicate with its single hop neighbors [37] without considering the relative importance of each neighboring node. However, depending on various factors, such as data distribution and graph topology, assigning weights to updates from different neighbors can be a very effective measure to improve convergence [36, 65]. This chapter introduced the weighted hybrid ADMM algorithms that takes into consideration the relative importance of different neighbors, manifested as edge weights. The efficacy of proposed algorithm is validated through numerical experiments.

Chapter 4 considers the influence of network topology over the performance of distributed optimization algorithms. Such a connection has been explored from the perspective of random walk [31, 30]. This chapter approach this problem through preconditioning, which is shown to be mathematically equivalent to assigning weights to edges that can formulated as a diagonal scaling of the unweighted problem [70]. Therefore, optimal edge weights can be found by searching for the best diagonal scaling matrix. Through relaxation, the later can be solved using semidefinite programming. The merits of proposed approach are numerically validated through experiments.

Chapter 5 turns to another key issues with distributed algorithms: asynchrony. All the approaches discussed in Chapter 2, 3, and 4 are synchronous, meaning synchronization processes need to be performed to maintain identical clock time for all nodes. The synchronization of clock can be a quite challenging problem itself [59, 80, 75], not to mention the additional cost and delay incurred by this process. Therefore, algorithms with reduced or no synchronization would be ideal in practical settings. To cope with this challenge, this chapter comes up with a

novel asynchronous hybrid ADMM algorithm [72] that forgoes the requirement of synchronization and allows the existence of multiple master nodes. The convergence of proposed algorithm is established under mild conditions, and its advantages are showcased by the numerical tests.

Finally, chapter 6 concludes the present thesis with some discussions of proposed methods and some future research directions.

1.3 Notational conventions

In the subsequent chapters, the following notation will be used throughout. Bold lowercase (uppercase) letters will be used for column vectors (matrices), while the normal ones for scalars (constants). Calligraphic symbols are reserved for sets. The $n \times n$ identity matrix is denoted by $\boldsymbol{I}_n$, and all-one vector by $\mathbf{1}$ and all-zero vector $\mathbf{0}$. The size of matrices (vectors) is omitted if it is obvious from the context; otherwise it is indicated by a subscript. Operator $(\cdot)^\top$ stands for matrix transpose, and $\lambda_{\max}(\cdot)$ and $\lambda_{\min}(\cdot)$ represent the largest and smallest eigenvalues, $|\cdot|$ the cardinality of a set, or the absolute value of a number. The l_1 norm is denoted by $\|\cdot\|_1$, l_∞ norm by $\|\cdot\|_\infty$, l_p norm by $\|\cdot\|_p$. The Kronecker product of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ is represented by $\boldsymbol{A} \otimes \boldsymbol{B}$.

Chapter 2

Hybrid ADMM: A Unifying and Fast Approach for Decentralized Optimization

2.1 Introduction

Recent advances in machine learning, signal processing and data mining, have led to important problems that can be formulated as distributed optimization over networks. Such problems entail parallel processing of data acquired by interconnected nodes and arise frequently in several applications, including data fusion and processing using sensor networks [116, 90, 66, 100]; vehicle coordination [98, 97]; power state estimation [57]; clustering [27]; classification [29]; regression [77]; filtering [90]; and demodulation [121, 3], to name a few. Among the candidate solvers for such problems, the *alternating direction method of multipliers* (ADMM) [5, 9] stands out as an efficient and easily implementable algorithm of choice that has attracted much interest in recent years [24, 43, 51, 19], thanks to its simplicity, fast convergence, and easily decomposable structure.

Many distributed optimization problems can be formulated in a consensus form, and solved efficiently by ADMM [9, 37]. The solver involves two basic steps: (i) a *communication step* for exchanging information with a central processing unit, the so-called *fusion center* (FC); and, (ii) an *update step* for updating the local variables at each node. By alternating between

the two, local iterates eventually converge to the global solution. This approach is referred to as *centralized* consensus ADMM (C-CADMM), and although it has been successfully applied in various settings, it may not always present the preferable solver. In large-scale systems for instance, the cost of connecting each node to the FC may become prohibitive as the overhead of communicating data to the FC may be overwhelming, and the related storage requirement could surpass the capacity of a single FC. Furthermore, having one dedicated FC can lead to a single point of failure. In addition, there might be privacy-related issues that restrict access to private data.

Decentralized optimization on the other hand, forgoes with the FC by exchanging information only among single-hop neighbors. As long as the network is connected, local iterates can consent to the globally optimal decision variable, thanks to the aforementioned information exchange. This method – referred to as *decentralized* consensus ADMM (D-CADMM) – has attracted considerable interest; see e.g., [37] for a review of applications in communications and networking. In large-scale networks, D-CADMM's convergence slows down as the per-node information experiences large delays to reach remote destinations through multiple neighbor-to-neighbor communications.

2.1.1 Our Contributions

To address the aforementioned limitations, the present chapter puts forth a novel decentralized framework, that we term hybrid consensus ADMM (H-CADMM), which unifies and markedly broadens C-CADMM and D-CADMM. Our contributions are in five directions:

(i) H-CADMM features *hybrid* updates accommodating communications with both the FCs and single-hop neighbors, thus bridging centralized with fully decentralized updates. This makes H-CADMM appealing for large-scale networks with multiple local FCs – a situation none of the existing approached is designed to handle.

(ii) A novel formulation of D-CADMM without duplicate constraints (dual variables commonly adopted by decentralized learning [57, 37, 103]) emerges simply by specializing the hybrid constraints to coincide with those arising from the purely neighborhood-based formulation.

(iii) Linear convergence is established, along with a rate bound, and specializes to C- and D-CADMM. The parameter setting to achieve the optimal bound is also provided.

(iv) H-CADMM is flexible to deploy FCs as needed to maximize performance gains, thus striking a desirable trade-off between the number of FCs deployed and convergence gain sought.

(v) The capability of handling hybrid constraints not only deals with mixed updates, but also effects "in-network acceleration" in decentralized operation without incurring noticeable increase in the overall complexity.

2.1.2 Related Work

Distributed optimization over networks has attracted much attention since the seminal works in [5, 110, 87], where gradient-based parallel algorithms were developed. Since then, several alternatives have been advocated, including subgradient methods [82, 83, 107, 109], stochastic subgradients [105], proximal gradient [102], dual averaging [21, 108, 52], random walk [76], splitting methods [117], gossip algorithms [20] and primal-dual methods [104].

The decomposability of CADMM makes it particularly well suited for distributed optimization. Along with its many variants, including centralized [5], decentralized [37, 103], weighted [66, 74] and Nesterov accelerated [43], CADMM has gained wide popularity.

ADMM was introduced in the 1970s [42], and its convergence analysis can be found in e.g. [32, 23]. Local linear convergence of ADMM for linear or quadratic programs is established in [6]; see also [51] where the cost is a sum of component costs. Global linear convergence of a more general form of ADMM is reported in [19, 64]; and linear convergence for a generalized formulation of consensus ADMM using the so-called "communication matrix" in [74].

Though D-CADMM has been applied to various problems [66, 121, 77, 100, 3, 4, 95, 34], its convergence remained open until linear convergence of D-CADMM was established in [103], and in [65] for its weighted counterpart. A successive orthogonal projection approach for distributed learning over networked nodes is introduced in [92], where nodes cannot communicate, but each node can access only limited amounts of data and agreement is enforced across nodes sharing the same data. A distributed ADMM algorithm that deals with node clusters was proposed, for which linear convergence was also substantiated [54]. However, it relies on the gossip algorithm for communication within clusters when cluster head is absent, and the explicit rate bound is difficult to obtain; and, it admits no closed-form representation for general networks. The present contribution is the first principled attempt to tackling the hybrid consensus problem.

2.1.3 Outline and Notation

The rest of the chapter is organized as follows. Section 2.2 states the problem and outlines two existing solvers, namely C-CADMM and D-CADMM; Section 2.3 develops H-CADMM, and shows its connections to both C-CADMM and D-CADMM; Section 2.4 establishes linear convergence of H-CADMM, and discusses parameter settings that can afford optimal performance; Section 2.5 introduces the notion of "in-network acceleration;" Section 2.6 reports the results of numerical tests; and Section 2.7 concludes this work.

2.2 Preliminaries

For a network of N nodes, consensus optimization amounts to solving problems of the form

$$\min_{\boldsymbol{x}} \quad \sum_{i=1}^{N} f_i(\boldsymbol{x}) \tag{2.1}$$

where $f_i(\cdot)$ is the i-th cost – only available to node i; and $\boldsymbol{x} \in \mathbb{R}^l$ is the common decision variable.

A common approach to solving such problems is to create a local copy of the global decision variable for each node, and impose equality constraints among all local copies; that is,

$$\begin{aligned} \min_{\{\boldsymbol{x}_i\}} \quad & \sum_{i=1}^{N} f_i(\boldsymbol{x}_i) \\ \text{s. to} \quad & \boldsymbol{x}_1 = \boldsymbol{x}_2 = \ldots = \boldsymbol{x}_N \end{aligned} \tag{2.2}$$

where $\{\boldsymbol{x}_i\}$ are the local copies, and equality is enforced to ensure equivalence of (2.1) with (2.2). As a result, the global decision variable in (2.1), is successfully decoupled to facilitate distributed processing.

Each node optimizes locally its component of cost, and the equality constraints are effected by exchanging information among nodes, subject to restrictions. Indeed, in the *centralized* case nodes communicate with a single FC, while in the fully *decentralized* case, nodes can only communicate with their immediate neighbors.

We model communication constraints in the decentralized setting as an undirected graph $\mathcal{G} := (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ collects node indices and $\mathcal{E}$ collects pairs of nodes that can communicate.

Each vertex $\{v_i\}$ corresponds to one node, and the presence of edge $(v_i, v_j) \in \mathcal{E}$ denotes that nodes i and j can communicate. With N(respectively M) denoting the number of nodes (edges), we will label nodes (edges) using the set $\{1, 2, \ldots, N\}$ (respectively $\{1, 2, \ldots, M\}$). We will further define the neighborhood set of node i as $\mathcal{N}_i := \{j|(v_i, v_j) \in \mathcal{E}\}$.

The following assumptions will be adopted about the graph and the local cost functions.

Assumption 1 (Connectivity)**.** Graph $\mathcal{G} := (\mathcal{V}, \mathcal{E})$ is connected.

Assumption 2 (Strong convexity)**.** Local cost f_i is σ_i-strongly convex; that is, for any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^l$,

$$f_i(\boldsymbol{y}) \geq f_i(\boldsymbol{x}) + \nabla^\top f_i(\boldsymbol{x})(\boldsymbol{y} - \boldsymbol{x}) + \frac{\sigma_i}{2}\|\boldsymbol{y} - \boldsymbol{x}\|_2^2.$$

Assumption 3 (Lipschitz continuous gradient)**.** Local f_i is differentiable, and has Lipschitz continuous gradient; that is, for any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^l$,

$$\|\nabla f_i(\boldsymbol{x}) - \nabla f_i(\boldsymbol{y})\|_2 \leq L_i\|\boldsymbol{x} - \boldsymbol{y}\|_2.$$

For brevity, we will henceforth focus on $l = 1$, but Appendix 2.8.1 outlines the generalization to $l \geq 2$.

2.2.1 Centralized consensus ADMM

With a centralized global (G)FC, consensus is guaranteed when each node forces its local decision variable to equal that of the GFC. In iterative algorithms, this is accomplished through the update of each local decision variable based on information exchanged with the GFC. As a result, (2.1) can be formulated as

$$\begin{aligned} \min_{\{x_i\}} \quad & \sum_{i=1}^{N} f_i(x_i) \\ \text{s. to} \quad & x_i = z \end{aligned} \tag{2.3}$$

where z represents the GFC's decision variable (state).

ADMM solver by (2.3) by (i) forming the augmented Lagrangian; and (ii) performing Gauss-Seidel updates of primal and dual variables. Attaching Lagrange multipliers $\{\lambda_i\}_{i=1}^N$ to the equality constraints, and augmenting the Lagrangian with the penalty parameter ρ, we

arrive at

$$L(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\lambda}) = F(\boldsymbol{x}) + \boldsymbol{\lambda}^\top(\boldsymbol{x} - z\mathbf{1}) + \frac{\rho}{2}\|\boldsymbol{x} - z\mathbf{1}\|_2^2 \tag{2.4}$$

where $\boldsymbol{x} := [x_1, x_2, \ldots, x_N]^\top$, $\boldsymbol{\lambda} := [\lambda_1, \lambda_2, \ldots, \lambda_N]^\top$, $F(\boldsymbol{x}) := \sum_{i=1}^N f_i(x_i)$, and $z = \bar{x} = N^{-1}\sum_{i=1}^N x_i$. Per entry (node) i, the ADMM updates are (see e.g. [9])

$$x_i^{k+1} = (\nabla f_i + I)^{-1}(\rho\bar{x}^k - \lambda_i^k) \tag{2.5a}$$

$$\lambda_i^{k+1} = \lambda_i^k + \rho(x_i^{k+1} - \bar{x}^k) \tag{2.5b}$$

where z has been eliminated, and the inverse in (2.5a) is a shorthand for x_i^{k+1} being the solution of

$$\nabla f_i(x_i^{k+1}) + x_i^{k+1} = (\rho\bar{x}^k - \lambda_i^k). \tag{2.6}$$

Specialized to (2.5) C-CADMM boils down to the following three-step updates.

C1. Node i solves (2.5a), and x_i^{k+1} to the GFC;

C2. The GFC updates its global decision variable by averaging local copies, and broadcasts the updated value z^{k+1} back to all nodes; and

C3. Node i updates its Lagrange multiplier as in (2.5b).

2.2.2 Decentralized consensus ADMM

In the decentralized setting, no GFC is present and nodes can only communicate with their one-hop neighbors. If the underlying graph $\mathcal{G}$ is connected, consensus constraints effect agreement across nodes.

Consider an auxiliary variable $\{z_{ij}\}$ per edge $(v_i, v_j) \in \mathcal{E}$, and re-write (2.2) in as

$$\begin{aligned} \min_{\{x_i\}} \quad & \sum_{i=1}^N f_i(x_i) \\ \text{s. to} \quad & x_i = z_{ij},\ x_j = z_{ji},\ (v_i, v_j) \in \mathcal{E}. \end{aligned} \tag{2.7}$$

For undirected graphs, we have $z_{ij} = z_{ji}$. With M edges in total, (2.7) includes $4M$ equality

constraints, that can be written in matrix-vector form, leading to the compact expression

$$\begin{aligned} \min_{\{x_i\}} \quad & F(\boldsymbol{x}) \\ \text{s. to} \quad & \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} = \boldsymbol{0} \end{aligned} \tag{2.8}$$

where $\boldsymbol{z}$ is the vector concatenating all $\{z_{ij}\}$ in arbitrary order, $\boldsymbol{A} := [\boldsymbol{A}_1^\top, \boldsymbol{A}_2^\top]^\top$ with $\boldsymbol{A}_1, \boldsymbol{A}_2 \in \mathbb{R}^{N \times 2M}$ defined such that if the q-th element of $\boldsymbol{z}$ is z_{ij}, then $(\boldsymbol{A}_1)_{qi} = 1$, $(\boldsymbol{A}_2)_{qj} = 1$, and all other elements are zeros; while $\boldsymbol{B} := [-\boldsymbol{I}_{2M}^\top, -\boldsymbol{I}_{2M}^\top]^\top$.

Formulation (2.8) is amenable to ADMM. To this end, one starts with the augmented Lagrangian

$$L(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\lambda}) = F(\boldsymbol{x}) + \boldsymbol{\lambda}^\top(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z}) + \frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z}\|_2^2 \tag{2.9}$$

where Lagrange multiplier vector $\boldsymbol{\lambda} := [\boldsymbol{\beta}^\top, \boldsymbol{\gamma}^\top]^\top$ is split in sub-vectors $\boldsymbol{\beta},\ \boldsymbol{\gamma} \in \mathbb{R}^{2M}$, initialized with $\boldsymbol{\beta}^0 = -\boldsymbol{\gamma}^0$. After simple manipulations one obtains the simplified ADMM updates (see [103] for details):

$$x_i^{k+1} = (\nabla f_i + \rho|\mathcal{N}_i|I)^{-1}\left[\frac{\rho}{2}\sum_{j\in\mathcal{N}_i}(x_i^k + x_j^k) - y_i^k\right] \tag{2.10a}$$

$$y_i^{k+1} = y_i^k + \frac{\rho}{2}\sum_{j\in\mathcal{N}_i}\left(x_i^{k+1} - x_j^{k+1}\right) \tag{2.10b}$$

where $\boldsymbol{y} := (\boldsymbol{A}_1 - \boldsymbol{A}_2)^\top\boldsymbol{\beta}$, and the inverse in (2.10a) is a shorthand for x_i^{k+1} being the solution of

$$\nabla f_i(x_i^{k+1}) + \rho|\mathcal{N}|x_i^{k+1} = (\rho/2)\sum_{j\in\mathcal{N}_i}(x_i^{k+1} + x_j^{k+1}). \tag{2.11}$$

The fact that the per-node updates in (2.10) involve only single-hop neighbors justifies the term decentralized consensus ADMM (D-CADMM).

In a nutshell, D-CADMM works as follows:

D1. Each node sends its local variable to all its single-hop neighbors;

D2. Upon receiving information from all its neighbors, node i updates its local variable as in (2.10a);

D3. Node x_i, node i updates its dual variable y_i as in (2.10b).

2.3 Hybrid consensus ADMM

Rather than a single GFC that is connected to all nodes, here we consider optimization over networks with *multiple LFCs*. Such a setup can arise in large-scale networks, where bandwidth, power, and computational limits or even security concerns may discourage deployment of a single GFC. These considerations prompt the deployment of multiple LFCs each of which communicates with a limited number of nodes. No prior ADMM-based solver can deal with this setup as none is capable of handling *hybrid constraints* that are present when nodes exchange information not only with LFCs but also with their single-hop neighbors. This section, introduces H-CADMM that is particularly designed to handle this situation.

2.3.1 Problem formulation

In contrast to the simple graph $\mathcal{G}$ used in the fully decentralized setting, we will employ *hypergrahs* to cope with hybrid constraints. A hypergraph is a tuple $\mathcal{H} := (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the vertex set and $\mathcal{E} := \{\mathcal{E}_i\}_{i=1}^{M}$ denotes the collection of hyperedges. Each $\mathcal{E}_i$ comprises a set of vertices, $\mathcal{E}_i \subset \mathcal{V}$ with cardinality $|\mathcal{E}_i| \geq 2, \ \forall i$. If $|\mathcal{E}_i| = 2$, then it reduces to a *simple* edge. A vertex v_i and an edge $\mathcal{E}_j$ are said to be incident if $v_i \in \mathcal{E}_j$. Hypergraphs are particularly suitable for modeling hybrid constraints because each LFC can be modeled as one hyperedge consisting of all its connected nodes.

With N nodes, M hyperedges, and their corresponding orderings, we can associate each edge variable z_j with hyperedge j. Then the hybrid constraints can be readily reparameterized as $x_i = z_j$, $\forall i : v_i \in \mathcal{E}_j$. Consider now vectors $\boldsymbol{x} \in \mathbb{R}^N$, $\boldsymbol{z} \in \mathbb{R}^M$ collecting all local $\{x_i, z_j\}$s, and matrices $\boldsymbol{A} \in \mathbb{R}^{T\times N}$, $\boldsymbol{B} \in \mathbb{R}^{T\times M}$ constructed to have nonzero entries $A_{ti} = 1$, $B_{tj} = 1$ corresponding to t-th constraint $x_i - z_j = 0$. For T equality constraints, the hybrid form of (2.1) can thus be written compactly as

$$\begin{aligned} \min_{x_i} \quad & \sum_{i=1}^{N} f_i(x_i) \\ \text{s.to} \quad & \boldsymbol{A}\boldsymbol{x} - \boldsymbol{B}\boldsymbol{z} = \boldsymbol{0}. \end{aligned} \tag{2.12}$$

Let now $\boldsymbol{C} \in \mathbb{R}^{N\times M}$ denote the incidence matrix of the hypergraph, formed with entries $C_{ij} = 1$ if node i and edge j are incident, and $C_{ij} = 0$ otherwise; d_i the degree of node i (the

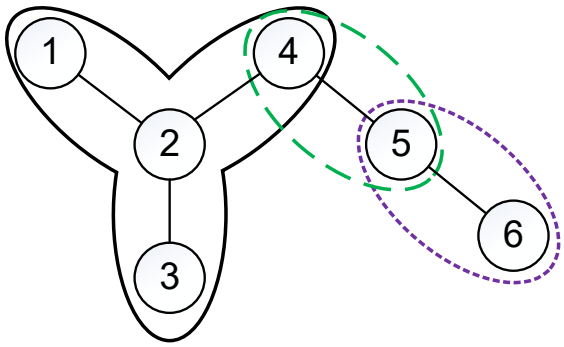


Figure 2.1: An example demonstrating how hyperedges are created. Both the underlying graph (simple edges in black straight lines) and the hypergraph (hyperedges as ellipsoids) are shown for comparison.

number of incident edges of node i); e_j the degree of hyperedge j (the number of incident nodes of hyperedge j); diagonal matrix $\boldsymbol{D} \in \mathbb{R}^{N\times N}$ the node degree matrix (formed with d_i as its i-th diagonal element); and likewise $\boldsymbol{E} \in \mathbb{R}^{M\times M}$ the *edge degree* matrix (formed with $e_j = |\mathcal{E}_j|$ as its j-th diagonal element). With these notational conventions, we prove in the Appendix the following.

Lemma 1. *Matrices* $\mathbf{A}$ *and* $\mathbf{B}$ *in* (2.12) *satisfy*

$$\boldsymbol{A}^\top \boldsymbol{A} = \boldsymbol{D} \tag{2.13a}$$

$$\boldsymbol{B}^\top \boldsymbol{B} = \boldsymbol{E} \tag{2.13b}$$

$$\boldsymbol{A}^\top \boldsymbol{B} = \boldsymbol{C}. \tag{2.13c}$$

Example Consider the graph in Figure 2.1. The underlying graph has 6 nodes and 5 simple edges. We create one hyperedge containing nodes 1 to 4, nodes 4 and 5, and nodes 5 and 6. As a result, the hypergraph has $N = 6$ nodes and $M = 3$ hyperedges.

Creating variables x_i for nodes i and z_j for hyperedge j, we obtain $T = 8$ constraints, namely,

$$\begin{aligned} x_1 &= z_1, \quad x_2 = z_1 \\ x_3 &= z_1, \quad x_4 = z_1 \\ x_4 &= z_2, \quad x_5 = z_2 \\ x_5 &= z_3, \quad x_6 = z_3. \end{aligned} \tag{2.14}$$

Accordingly, one can construct matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ as

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

One can easily verify that each row of $\boldsymbol{A}\boldsymbol{x} - \boldsymbol{B}\boldsymbol{z} = \boldsymbol{0}$ represents one constraint and Lemma 1 indeed holds.

2.3.2 Algorithm

The augmented Lagrangian for (2.12) is

$$L(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\lambda}) = \sum_{i=1}^{N} f_i(x_i) + \boldsymbol{\lambda}^\top (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{B}\boldsymbol{z}) + \frac{\rho}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{B}\boldsymbol{z}\|_2^2 \tag{2.15}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^T$ collects all the Lagrange multipliers, and ρ is a hyper-parameter controlling the effect of the quadratic regularizer. ADMM updates can be obtained by cyclically solving for $\boldsymbol{x}$, $\boldsymbol{z}$ and $\boldsymbol{\lambda}$ the equations

$$\nabla f(\boldsymbol{x}^{k+1}) + \boldsymbol{A}^\top \boldsymbol{\lambda}^k + \rho \boldsymbol{A}^\top (\boldsymbol{A}\boldsymbol{x}^{k+1} - \boldsymbol{B}\boldsymbol{z}^k) = \boldsymbol{0} \tag{2.16a}$$

$$\boldsymbol{B}^\top \boldsymbol{\lambda}^k + \rho \boldsymbol{B}^\top (\boldsymbol{A}\boldsymbol{x}^{k+1} - \boldsymbol{B}\boldsymbol{z}^{k+1}) = \boldsymbol{0} \tag{2.16b}$$

$$\boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k + \rho(\boldsymbol{A}\boldsymbol{x}^{k+1} - \boldsymbol{B}\boldsymbol{z}^{k+1}). \tag{2.16c}$$

Equations (2.16) can be simplified by left-multiplying (2.16c) by $\boldsymbol{B}^\top$ and adding it to (2.16b) to obtain

$$\boldsymbol{B}^\top \boldsymbol{\lambda}^{k+1} = \boldsymbol{0}. \tag{2.17}$$

If $\boldsymbol{\lambda}$ is initialized such that $\boldsymbol{B}^\top \boldsymbol{\lambda}^0 = \boldsymbol{0}$, then $\boldsymbol{B}^\top \boldsymbol{\lambda}^k = \boldsymbol{0}$ for all $k \geq 0$. Eliminating $\boldsymbol{B}^\top \boldsymbol{\lambda}^k$

from (2.16b), one can solve for $\boldsymbol{z}$, and arrive at the closed form

$$\boldsymbol{z}^{k+1} = \boldsymbol{E}^{-1}\boldsymbol{C}^{\top}\boldsymbol{x}^{k+1}. \tag{2.18}$$

Similarly, by left-multiplying (2.16c) by $\boldsymbol{A}^{\top}$ and letting $\boldsymbol{y}^k := \boldsymbol{A}^{\top}\boldsymbol{\lambda}^k$, one finds

$$\boldsymbol{y}^{k+1} - \boldsymbol{y}^k = \rho(\boldsymbol{D}\boldsymbol{x}^{k+1} - \boldsymbol{C}\boldsymbol{z}^{k+1}). \tag{2.19}$$

Then simply plugging (2.18) into (2.16a) yields

$$\boldsymbol{x}^{k+1} = (\nabla f + \rho\boldsymbol{D}I)^{-1}(\rho\boldsymbol{C}\boldsymbol{z}^k - \boldsymbol{y}^k). \tag{2.20}$$

Recursions (2.18)–(2.20) summarize our H-CADMM, and their per-node forms are listed in Algorithm 1.

Remark 1. Two interesting observations are in order:

(i) Regardless of the number of attached nodes, each hyperedge serves as an LFC; and

(ii) Each LFC performs local averaging. Indeed, the entry-wise update of (2.18) shows that each hyperedge satisfies $z_i = (1/E_{ii})\sum_{j\in\mathcal{N}_i} x_j$. Hence, all hyperedges are treated equally in the sense that they are updated by the average value of all incident nodes.

2.3.3 Key relations

Here we unveil a relationship satisfied by the iterates $\{\boldsymbol{x}^k\}$ generated by Algorithm 1, which not only provides a different view of H-CADMM, but also serves as the starting point for establishing the convergence results in Section 2.4. This relation shows that $\boldsymbol{x}^{k+1}$ depends solely on the gradient of the local cost function, as well as the past $\{\boldsymbol{x}^k, \boldsymbol{x}^{k-1}, \ldots, \boldsymbol{x}^0\}$ that is also not dependent on the variables $\boldsymbol{z}^k$ and $\boldsymbol{y}^k$.

Lemma 2. *The sequence $\{\boldsymbol{x}^k\}$ generated by Algorithm 1 satisfies*

$$\boldsymbol{x}^{k+1} = -\frac{1}{\rho}\boldsymbol{D}^{-1}\nabla F(\boldsymbol{x}^{k+1}) + \boldsymbol{D}^{-1}\boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^{\top}\boldsymbol{x}^k - \boldsymbol{D}^{-1}(\boldsymbol{D} - \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^{\top})\sum_{t=0}^{k}\boldsymbol{x}^t. \tag{2.21}$$

Algorithm 1: Hybrid Consensus ADMM

Input: ρ, $\boldsymbol{x}^0$, $\boldsymbol{z}^0$, $\boldsymbol{y}^0 = \mathbf{0}$

while *stopping criterion not satisfied* **do**

  for $i = 1, \ldots, N$ **do**

    node i updates x_i^{k+1} by solving $\nabla f_i(x_i^{k+1}) + \rho|\mathcal{N}_i|x_i^{k+1} = \rho \sum_{j\in\mathcal{N}_i} z_j^k - y_i^k$

    send x_i^{k+1} to all incident FCs and neighbors

  for $j = 1, \ldots, M$ **do**

    FC j updates $z_j^{k+1} = \frac{1}{E_{jj}} \sum_{i\in\mathcal{N}_j} x_i^{k+1}$

    send z_j^{k+1} to all incident nodes

  for $i = 1, \ldots, N$ **do**

    node i updates $y_i^{k+1} = y_i^k + \rho\Big(D_{ii}x_i^{k+1} - \sum_{j\in\mathcal{N}_i} z_j^{k+1}\Big)$

Proof. Substituting (2.18) into (2.19), we obtain

$$\boldsymbol{y}^{k+1} - \boldsymbol{y}^k = \rho(\boldsymbol{D} - \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top)\boldsymbol{x}^k \tag{2.22}$$

which upon initializing with $\boldsymbol{y}^0 = \mathbf{0}$, leads to

$$\boldsymbol{y}^k = \rho(\boldsymbol{D} - \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top)\sum_{t=0}^{k} \boldsymbol{x}^t. \tag{2.23}$$

Plugging (2.18) and (2.23) into (2.20), yields

$$\nabla F(\boldsymbol{x}^{k+1}) + \rho\boldsymbol{D}\boldsymbol{x}^{k+1} = \rho\boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top\boldsymbol{x}^k - \rho(\boldsymbol{D} - \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top)\sum_{t=0}^{k} \boldsymbol{x}^t \tag{2.24}$$

from which we can readily solve for $\boldsymbol{x}^{k+1}$. □

Lemma 2 shows that x_i^{k+1} is determined by its past $\{x_i^t\}_{t=0}^k$, and the local gradient, namely $\nabla f_i(x_i^{k+1})$. This suggests a new update scheme, where each node maintains not only its current x_i^k but also $\sum_{t=0}^k x_i^t$.

Among the things worth stressing in Lemma 2 is the appearance of matrices $\boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top$ and $\boldsymbol{D} - \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top$. Since both play key roles in studying the evolution of (2.24), it is important to understand their properties and impact on the performance of the algorithm.

Lemma 3. *Matrices $\boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top$ and $\boldsymbol{D}-\boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top$ are positive semidefinite (PSD), and satisfy*

$$(\boldsymbol{D}-\boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top)\mathbf{1}=\mathbf{0}. \tag{2.25}$$

Proof. See Appendix. □

2.3.4 H-CADMM links to C-CADMM and D-CADMM

Modeling hybrid communication constraints as a hypergraph not only affords the flexibility to accommodate multiple LFCs, but also provides a unified view of consensus-based ADMM. Indeed, it is not difficult to show that by specializing the hypergraph, our proposed approach subsumes both centralized and decentralized consensus ADMM.

Proposition 1. *H-CADMM reduces to C-CADMM when there is only one hyperedge capturing all nodes.*

Proof. When there is a single hyperedge comprising all network nodes, we have $\boldsymbol{A}=\boldsymbol{I}_N$, and $\boldsymbol{B}=\mathbf{1}$. Using (2.13), we thus obtain

$$\boldsymbol{D}=\boldsymbol{A}^\top\boldsymbol{A}=\boldsymbol{I}_N \tag{2.26a}$$

$$\boldsymbol{E}=\boldsymbol{B}^\top\boldsymbol{B}=N \tag{2.26b}$$

$$\boldsymbol{C}=\boldsymbol{A}^\top\boldsymbol{B}=\mathbf{1}. \tag{2.26c}$$

Then, the update (2.18) at the GFC reduces to

$$z^{k+1}=\boldsymbol{E}^{-1}\boldsymbol{C}^\top\boldsymbol{x}^k=\frac{1}{N}\sum_{i=1}^{N}x_i^k=\bar{x}^k.$$

Similarly, (2.20) and (2.19) specialize to

$$\boldsymbol{x}^{k+1}=(\nabla f+\rho I)^{-1}(\rho\bar{x}^k\mathbf{1}-\boldsymbol{\lambda}^k) \tag{2.27a}$$

$$\boldsymbol{\lambda}^{k+1}=\boldsymbol{\lambda}^k+\rho(\boldsymbol{x}^{k+1}-\bar{x}^{k+1}\mathbf{1}) \tag{2.27b}$$

where we have used that $\boldsymbol{y}^k=\boldsymbol{A}^\top\boldsymbol{\lambda}^k=\boldsymbol{\lambda}^k$.

Comparing (2.27) with (2.5), it is not difficult to see that (2.5a) is just the entry-wise form

of (2.27a); and likewise for (2.27b) and (2.5b). Therefore C-CADMM can be viewed as a special case of H-CADMM with one hyperedge connecting all nodes. □

Proposition 2. *H-CADMM reduces to D-CADMM when every edge is a hyperedge.*

Proof. When each simple edge is modeled as a hyperedge, the resulting hyperedges will end up having degree 2, that is, $\boldsymbol{E} = 2\boldsymbol{I}$. Thus, (2.18) becomes

$$\boldsymbol{z}^{k+1} = \frac{1}{2}\boldsymbol{C}^\top \boldsymbol{x}^{k+1}. \tag{2.28}$$

Using (2.28), and eliminating $\boldsymbol{z}$ from (2.20) and (2.19) yields

$$\boldsymbol{x}^{k+1} = (\nabla f + \rho \boldsymbol{D} I)^{-1}(\frac{\rho}{2}\boldsymbol{C}\boldsymbol{C}^\top \boldsymbol{x}^k - \boldsymbol{y}^k) \tag{2.29a}$$

$$\boldsymbol{y}^{k+1} = \boldsymbol{y}^k + \rho(\boldsymbol{D} \ - \frac{1}{2}\boldsymbol{C}\boldsymbol{C}^\top)\boldsymbol{x}^{k+1}. \tag{2.29b}$$

To relate $\sum_{j\in\mathcal{N}}(x_i + x_j)$ in (2.10b) to (2.29b), notice that $D_{ii} = |\mathcal{N}_i|$ and $(\boldsymbol{C}\boldsymbol{C}^\top \boldsymbol{x})_i = \sum_{j\in\mathcal{N}_i}(x_i + x_j)$. Hence, it is straightforward to see that (2.10a) and (2.10b) are just the entry-wise versions of (2.29a) and (2.29b). Therefore, D-CADMM is also a special case of H-CADMM with each simple edge viewed as a hyperedge. □

Remark 2. In past works of fully decentralized consensus ADMM [77, 121, 37, 103], one edge is often associated with two variables $z_{ij},\ z_{ji}$ in order to decouple the equality constraint $x_i = x_j$, and express it as $x_i = z_{ij},\ x_j = z_{ji}$. Although eventually the duplicate variables are shown equal (and therefore discarded), their sheer presence leads to duplicate Lagrange multipliers, which can complicate the algorithm. Proposition 2 suggests a novel derivation of D-CADMM with only one variable attached to each edge, a property that can simplify the whole process considerably.

Example 1. Consider the simple graph depicted in Fig. 2.1 with 6 nodes and 5 edges. All nodes work collectively to minimize some separable cost. Solid lines denote the undirected graph connectivity, while ellipsoids represent hyperedges in the modeling hypergraph of H-CADMM. Let us consider C-CADMM, D-CADMM and H-CADMM solvers on this setting. To gain insight, we examine closely the update rules at node 4 that we list in Table 2.1.

As C-CADMM relies on a GFC connecting all nodes, every node receives updates from the GFC. This is demonstrated in the first row of Table 2.1.

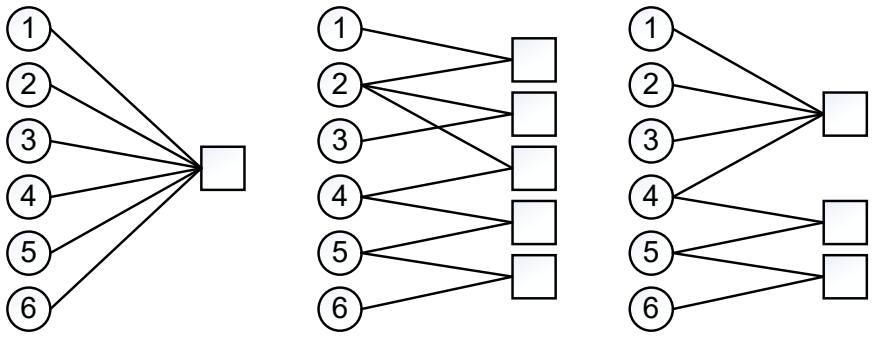


Figure 2.2: Communication graphs of C-CADMM, D-CADMM and H-CADMM for Example 1. Circles represent nodes, while squares represent hyperedges. Solid lines between nodes and hyperedges indicate nodes belonging to hyperedges.

D-CADMM on the other hand, allows communication only along edges (solid line). Thus, each node can only receive updates from its single-hop neighbors. Specifically, node 4 can only receive information from nodes 2 and 5, as can be seen in the second row of Table 2.1.

H-CADMM lies somewhere in between. It allows deployment of multiple LFCs, each of which is connected to a subset of nodes. The hypergraph model is shown in Figure 2.1 with hyperedges marked by ellipsoids. The union of 3 solid ellipsoids corresponds to the LFC, while dash and dotted ones represent the edges between nodes 4, 5, and 6. Speaking of node 4, the information needed to update its local variables comes from its neighbors, the LFC represented by $\sum_{i=1}^{4} x_i^{k+1}/4$; and node 5 represented by $(x_4^{k+1} + x_5^{k+1})/2$. That is exactly what we see in the third row of Table 2.1.

Remark 3. For all three consensus ADMM algorithms, the *fusion centers* – both global and local – act as averaging operators that compute, store and broadcast the mean values of the local estimates from all connected nodes.

Remark 4. The dual update per node acts as an accumulator forming the sum of residuals between the node and its connected *fusion centers*. With $\boldsymbol{L} := 2(\boldsymbol{D} - \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^{\top})$ denoting the Laplacian of the hypergraph, the dual update per node boils down to

$$\boldsymbol{y}^{k+1} = \boldsymbol{y}^{k} + \frac{\rho}{2}\boldsymbol{L}\boldsymbol{x}^{k+1}, \quad \forall k \geq 1.$$

Table 2.1: Comparison of ADMM update rules at node 4 using C-CADMM, D-CADMM and H-CADMM to solve the problem in Example 1

Method	Update Rules
C-CADMM	$x_4^{k+1} = \arg\min_{x_4} \left[f_4(x_4) + \lambda_4^k (x_4 - \bar{x}^k) + \frac{\rho}{2} \|x_4 - \bar{x}^k\|_2^2 \right]$ $\lambda_4^{k+1} = \lambda_4^k + \rho(x_4^{k+1} - \bar{x}^{k+1})$
D-CADMM	$x_4^{k+1} = (\nabla f_4 + 2\rho I)^{-1} \left(\rho x_4^k + \frac{\rho}{2}(x_2^k + x_5^k) - y_4^k \right)$ $y_4^{k+1} = y_4^k + \frac{\rho}{2} \left(2x_4^{k+1} - x_2^{k+1} - x_5^{k+1} \right)$
H-CADMM	$x_4^{k+1} = \arg\min_{x_4} \left[f_4(x_4) + y_4^k x_4 + \frac{\rho}{2} \left(\|x_4 - \frac{1}{4}\sum_{i=1}^{4} x_i^k\|^2 + \|x_4 - \frac{1}{2}\sum_{i=1}^{2} x_i^k\|^2 \right) \right]$ $y_4^{k+1} = y_4^k + \rho \left(2x_4^{k+1} - \frac{1}{4}\sum_{i=1}^{4} x_i^{k+1} - \frac{1}{2}\sum_{i=4}^{5} x_i^{k+1} \right)$

If dual variables are initialized such that $\boldsymbol{y}^0 = \mathbf{0}$, then

$$\boldsymbol{y}^k = \frac{\rho}{2} \boldsymbol{L} \sum_{t=1}^{k} \boldsymbol{x}^t.$$

This observation holds for all three algorithms.

Remark 5. Figure 2.2 exemplifies that H-CADMM "lies" somewhere between C-CADMM and D-CADMM. Clearly, consensus is attainable if and only if the communication graph is connected.

2.4 Convergence rate analysis

In this section, we analyze the convergence behavior of the novel H-CADMM algorithm. In particular, our main theorem establishes linear convergence and provides a bound on the rate of convergence, which depends on properties of both the objective function, as well as the underlying graph topology.

Apart from the assumptions made in Section 2.2, here we also need an additional one:

Assumption 4. There exists at least one saddle point $(\boldsymbol{x}^\star, \boldsymbol{z}^\star, \boldsymbol{y}^\star)$ of Algorithm 1 that satisfies the KKT conditions:

$$\nabla f(\boldsymbol{x}^\star) + \boldsymbol{y}^\star = \boldsymbol{0} \tag{2.30a}$$

$$\boldsymbol{z}^\star - \boldsymbol{E}^{-1}\boldsymbol{B}^\top \boldsymbol{A}\boldsymbol{x}^\star = \boldsymbol{0} \tag{2.30b}$$

$$(\boldsymbol{I} - \boldsymbol{B}\boldsymbol{E}^{-1}\boldsymbol{B}^\top)\boldsymbol{A}\boldsymbol{x}^\star = \boldsymbol{0}. \tag{2.30c}$$

This assumption is required for the development of Algorithm 1 as well as for the analysis of its convergence rate. If (as4) does not hold, either the original problem is unsolvable, or, it entails unbounded subproblems, or, a diverging sequence of $\boldsymbol{\lambda}^k$ [19].

Assumptions 1–4 guarantee the existence of at least one optimal solution. In fact, we can further prove that any saddle point is actually the unique solution of the KKT conditions (2.30), and hence of Algorithm 1.

Lemma 4. *If $\boldsymbol{\lambda}$ is initialized so that $\boldsymbol{B}^\top \boldsymbol{\lambda}^0 = \boldsymbol{0}$, and (as1)–(as4) hold, then $(\boldsymbol{x}^\star, \boldsymbol{z}^\star, \boldsymbol{y}^\star)$ is the* unique *optimal solution of* (2.30).

Proof. See Appendix. □

2.4.1 Linear rate of convergence

Alternating direction methods, including ADMM, have been thoroughly investigated in [19]. Similar to D-CADMM [103], conditions for establishing linear convergence rate in [19] are not necessarily satisfied by the H-CADMM setup[1]. Therefore, we cannot establish linear convergence rate simply by reformulating it as a special case of existing ADMM approaches.

One way to overcome this obstacle is to adopt a technique similar to [103], as we did in [71], to obtain a relatively loose bound on convergence, in the sense that it could not capture significant accelerations observed in practice by varying the topology of the underlying graph. For strongly convex costs, a tighter bound has been reported recently [74]. However, H-CADMM is not amenable to the analysis in [74] since the linear constraint coefficients $\boldsymbol{A}$ and $\boldsymbol{B}$ cannot be recovered from the communication matrix.

[1]This should be expected since H-CADMM reduces to D-CADMM upon modeling each simple edge as an hyperedge.

We establish convergence by measuring the progress in terms of the G-norm i.e. the semi-norm defined by $\|\boldsymbol{x}\|_G := \boldsymbol{x}^\top \boldsymbol{G}\boldsymbol{x}$, where $\boldsymbol{G}$ is the PSD matrix,

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top \end{bmatrix}. \tag{2.31}$$

The G-norm is properly defined since both $\boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top$ and $\boldsymbol{D} - \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top$ are PSD (see Lemma 3). Consider now the square root, $\boldsymbol{Q} := (\boldsymbol{D} - \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top)^{1/2}$, and define two auxiliary sequences

$$\boldsymbol{r}^k := \sum_{t=0}^{k} \boldsymbol{Q}\boldsymbol{x}^t, \quad \boldsymbol{q}^k := \begin{bmatrix} \boldsymbol{r}^k \\ \boldsymbol{x}^k \end{bmatrix}. \tag{2.32}$$

These two sequences play an important role in establishing linear convergence of the proposed algorithm. Before we establish such a convergence result we first need to bound the gradient of $F(\cdot)$.

Lemma 5. *If (as1)–(as4) hold, then for any $k \geq 0$, we have*

$$\boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top(\boldsymbol{x}^{k+1} - \boldsymbol{x}^k) = -\boldsymbol{Q}(\boldsymbol{r}^{k+1} - \boldsymbol{r}^\star) - \frac{1}{\rho}\left(\nabla F(\boldsymbol{x}^{k+1}) - \nabla F(\boldsymbol{x}^\star)\right). \tag{2.33}$$

Proof. See Appendix. □

Let $\lambda_{\max} := \lambda_N(\boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top)$ denote the largest eigenvalue of $\boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top$ and $\lambda_{\min} := \lambda_2(\boldsymbol{D} - \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top)$ the second smallest eigenvalue of $\boldsymbol{D} - \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top$.

Theorem 1. *Under (as1)–(as4) for any $\rho > 0$, $\beta \in (0, 1)$, and $k > 0$, H-CADMM iterates in (8) satisfy*

$$\|\boldsymbol{x}^k - \boldsymbol{x}^\star\|_{\boldsymbol{G}}^2 \leq \left(\frac{1}{1+\delta}\right)^k \|\boldsymbol{q}^0 - \boldsymbol{q}^\star\|_{\boldsymbol{G}}^2 \tag{2.34}$$

where $\boldsymbol{G}$ and $\boldsymbol{q}$ as in (26) and (27), and δ satisfies

$$\delta \leq \min\left\{\frac{2\beta\sigma}{\rho(\lambda_{max} + \frac{2\lambda_{max}}{\lambda_{min}})}, \frac{(1-\beta)\rho\lambda_{min}}{L}\right\}. \tag{2.35}$$

Proof. See Appendix. □

Theorem 1 asserts that $\boldsymbol{x}^k$ converges linearly to the optimal solution $\boldsymbol{x}^\star$ at a rate bounded by $1/(1+\delta)$. Larger δ implies faster convergence.

Note that while Theorem 1 is proved for $l = 1$, it can be generalized to $l \geq 2$ (see (2.45) in Appendix 2.8.1).

Remark 6. Assumptions 2 and 3 are sufficient conditions for establishing linear convergence rate of H-CADMM.

2.4.2 Fine-tuning the parameters

Theorem 1 characterizes the convergence of iterates generated by H-CADMM. Parameter δ is determined by the local costs, the underlying communication graph topology, and the scalar ρ. By tuning these parameters, one can maximize the convergence bound, to speed up convergence in practice too. With local costs and the graph fixed, one can maximize δ by tuning ρ. When possible to choose the number and locations of LFCs, we can effectively alter the topology of the communication graph – hence modify $\lambda_{\max}$ and $\lambda_{\min}$ – to improve convergence.

Theorem 2. *Under assumptions 1–4 the optimal convergence rate bound*

$$\delta \leq \frac{1}{\sqrt{\frac{L}{\sigma}\frac{\lambda_{max}}{\lambda_{min}}(1+2\frac{\lambda_{max}}{\lambda_{min}})}} \tag{2.36}$$

is achieved by setting

$$\rho = \sqrt{\frac{2\sigma L}{\lambda_{max}\lambda_{min}(1+\frac{\lambda_{max}}{\lambda_{min}})}}. \tag{2.37}$$

Proof. The optimal $\beta^\star \in (0,1)$ maximizing δ in Theorem 1 is

$$\beta^\star = \frac{\rho^2\lambda_{\max}\lambda_{\min}(1+2\frac{\lambda_{\max}}{\lambda_{\min}})}{2\sigma L+\rho^2\lambda_{\max}\lambda_{\min}(1+2\frac{\lambda_{\max}}{\lambda_{\min}})} \tag{2.38}$$

and is obtained by equating the two terms in (2.35)

$$\frac{2\beta\sigma}{\rho(\lambda_{\max}+2\frac{\lambda_{\max}}{\lambda_{\min}})} = \frac{(1-\beta)\rho\lambda_{\min}}{L}. \tag{2.39}$$

Substituting (23) into (34), one arrives at

$$\delta = \frac{2\sigma\rho\lambda_{\min}}{2\sigma L+\rho^2\lambda_{\max}\lambda_{\min}(1+2\frac{\lambda_{\max}}{\lambda_{\min}})}. \tag{2.40}$$

Maximizing δ by varying ρ eventually leads to (2.36), with the optimal

$$\rho^{\star} = \sqrt{\frac{2\sigma L}{\lambda_{\max}\lambda_{\min}(1+2\frac{\lambda_{\max}}{\lambda_{\min}})}}. \tag{2.41}$$

□

Upon defining the cost condition number as

$$\kappa_F := \max_i \frac{L_i}{\sigma_i}$$

and the graph condition number as

$$\kappa_G := \frac{\lambda_N(\boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^{\top})}{\lambda_2(\boldsymbol{D}-\boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^{\top})} \tag{2.42}$$

the optimal convergence rate can be bounded as

$$\delta \leq \frac{1}{\sqrt{\kappa_F\kappa_G(1+2\kappa_G)}}. \tag{2.43}$$

Clearly, the bound in (38) is a decreasing function of κ_F and κ_G. Therefore, decreasing both will drive the bound larger, possibly resulting in a faster rate of convergence. On the one hand, smaller cost condition number makes the cost easier to optimize; on the other hand, smaller graph condition number implies improved connectivity. Indeed, when the communication hypergraph is just a simple graph – a case for which H-CADMM reduces to D-CADMM with $\boldsymbol{D}-\boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^{\top} = \mathbf{L}/2$, then $\lambda_{\min}$ is the smallest nonzero eigenvalue of the Laplacian, which is related to bottlenecks in the underlying graph [18].

Remark 7. Theorem 2 shows that the number of iterations it takes to achieve an ϵ-accurate solution is $\mathcal{O}(\sqrt{\kappa_F}\log(\frac{1}{\epsilon}))$. The dependence on $1/\sqrt{\kappa_F}$ improves over [103], which had established a dependence of $1/\kappa_F^2$.

2.5 Graph-aware acceleration

Distributed optimization over networks using a central GFCs is not feasible for several reasons including communication constraints and privacy concerns. At the other end of the spectrum,

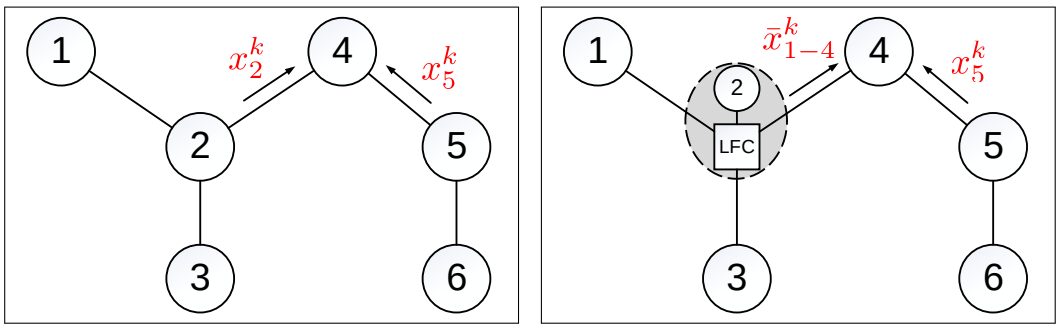


Figure 2.3: A demonstration of in-network acceleration applied to the problem of Example 1. The shaded dashed circle in (b) is equivalent to node 2 in (a), except that a virtual FC (square LFC) is created logically, making it amenable to the application of H-CADMM. The interface to other nodes remains the same. The information exchanged through the edges changes. For example, the message sent from node 2 to 4 changes from x_2^k, which describes only the state of node 2, to $x_{1-4}^k = 1/N \sum_{n=1}^4 x_n^k$ which contains information about 4 nodes.

decentralized schemes relying on single-hop communications may suffer from slow convergence, especially when the network has a large diameter or bottlenecks. H-CADMM fills the gap by compromising between the two aforementioned extremes. By carefully deploying multiple LFCs, it becomes possible to achieve significant performance gains whilst abiding by cost and privacy constraints.

In certain cases, leveraging the topology of the LFCs deployed could bring sufficient gains. Instead of, or complementing gains from these *actual* LFCs, this section advocates that gains in H-CADMM convergence are possible through *virtual FCs* on judiciously selected nodes. We refer to the benefit brought by virtual FCs as *in-network acceleration* (see Figure 2.3 for a simple illustration); it will be confirmed by numerical tests, virtual FCs can afford a boost in performance "almost for free"; simply by exploiting the actual network topology.

The merits of in-network acceleration through virtual FCs at a subset of selected nodes (*hosts*), can be recognized in the following four aspects.

- *Hardware*. Relying on virtual LFCs, in-network acceleration requires no modifications in the actual topology and hardware.

- *Interface*. The other nodes "see" exactly the same number of neighbors, so there is no change in the communication interface. However, the information exchanged is indeed

different.

- *Computational complexity*. Except for the host nodes, the update rules for both primal and dual variables remain the same. Each host however, serves a dual role: as an FC, as well as an ordinary node. We know from Algorithm 1 that the computations performed per LFC involve averaging information from all connected nodes, which is simple compared to updating the local variables. Thus, the computational complexity remains of the same order, while the total computational cost decreases as less iterations are necessary to reach a target level of accuracy.

- *Communication cost*. Since there is no change of the communication interface, the communication cost remains invariant. Once again, the total communication cost can further drop, since H-CADMM enjoys faster convergence.

Given that the interface does not change, nor extra communication/computation cost is incurred, one can think of in-network acceleration as a sort of "free lunch" approach, with particularly attractive practical implications.

Remark 8. The communication cost per iteration refers to the total number of transmission needed for updating each variable once, hence the total communication cost. In the graph of Figure 2.3, the nodes need to transmit 2×5 times to finish one iteration (each node must send and receive once to update its variables and there are 5 edges), so the communication cost per iteration is 10 transmission.

2.5.1 Strategies for selecting the local FCs

A reasonable question to ask at this point is: *"How should one select the nodes to host the virtual FCs?"* Unfortunately, there is no simple answer. The question would have been easier if we could choose as many LFCs as necessary to achieve the maximum possible acceleration. In practice however, we do not always have the luxury to place as many virtual LFCs as we want, for reasons that include lack of control over some nodes, and difficulty to modify internal updating rules. And even if we could, picking the right nodes to host the LFCs under a general network architecture might not be straightforward.

A reasonable way forth would be to maximize the convergence rate bound, δ, subject to a maximum number of LFCs, hoping that the optimal solution would yield the best rate of

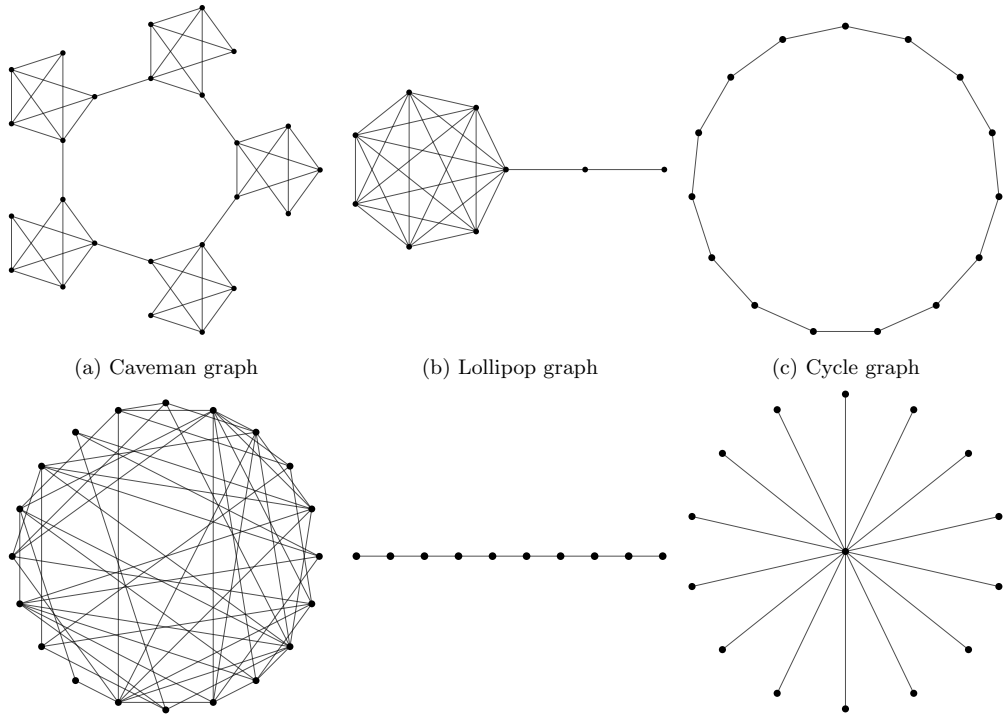
Algorithm 2: Greedy Selection of LFCs

Input: LFC budget B, graph $\mathcal{G} := (\mathcal{V}, \mathcal{E})$
$\mathcal{H} \leftarrow$ empty set
$c \leftarrow 0$
while $\mathcal{V}$ *not empty* **do**
  mark $v \in \mathcal{V}$ with largest node degree as a LFC
  add hyperedge $\{v\} \cup \mathcal{N}_v$ to $\mathcal{H}$
  delete $\{v\} \cup \mathcal{N}_v$ from $\mathcal{V}$
  delete $\{(v_1, v_2) | v_1, v_2 \in \{v\} \cup \mathcal{N}_v\}$ from $\mathcal{E}$
  $c \leftarrow c + 1$
  if $c \geq B$ **then**
    break
$\mathcal{H} \leftarrow \mathcal{H} \cup \mathcal{E}$
generate $\boldsymbol{C}$ from $\mathcal{H}$
Output: incidence matrix $\boldsymbol{C}$

convergence in practice. However, this turns out to involve optimizing the ratio of eigenvalues, which is typically difficult to solve. For this reason, we will resort to heuristic methods.

Intuitively, one may choose the nodes with highest degree so as to maximize the effect of virtual LFCs. However, one should be careful when applying this simple approach to clustered graphs. For example, consider the graph consisting of two cliques (connected by a short path), comprising n_1 and n_2 nodes, respectively. Each node in the larger clique has higher degree than every node in the smaller one. As a result, always assigning the role of LFC to the largest degree nodes would disregard the nodes of the smaller clique (when our budget is less than n_1), while one could apparently take care of both cliques with as few as two LFCs. Taking this into account, we advocate a greedy LFC selection (Algorithm 2), which prohibits placing virtual LFCs within the neighborhood of other FCs.

Remark 9. For simplicity, Algorithm 2 relies on degree information to select LFCs. More elaborate strategies would involve richer structural properties of the underlying topology, to identify more promising nodes at the expense of possibly computationally heavier LFC selection. In general, LFC selection strategies offer the potential of substantially increasing the convergence rate over random assignment. Note however, that regardless of the choice of virtual LFCs, H-CADMM remains operational.



(a) Caveman graph (b) Lollipop graph (c) Cycle graph

(d) Erdős Rényi graph (e) Line graph (f) Star graph

Figure 2.4: Plots of graphs used in numerical tests. For clarity, we keep the number of nodes limited, and in case of random graphs we show typical realizations.

2.5.2 On H-CADMM's "free lunch"

In-network acceleration has several advantages, allowing for faster convergence essentially "without paying any price." At first glance, this appears to be a "free lunch" type of benefit, and deservedly makes one skeptical. Actually, the benefit comes from leveraging information that is completely overlooked by fully decentralized methods. To see this, recall that in D-CADMM, each node communicates with only one neighbor each time, without accounting for the entire neighborhood. Instead, H-CADMM manages to exploit network topology by creating virtual LFCs that gather and share information with the whole neighborhood. It is this additional information that enables faster flow of data, and hence faster convergence. Therefore, it is not a "free lunch" for H-CADMM; but a lunch "not even tasted" by D-CADMM.

2.6 Numerical tests

In this section, we test numerically the performance of H-CADMM, and also validate our analytical findings.

2.6.1 Experimental settings

Throughout this section, we consider several interconnected nodes trying to estimate one value, x_0, based on local observations $o_i = x_0 + \epsilon_i$, where $\epsilon_i \sim \mathcal{N}(0, 0.1)$ is the measurement noise. This can be solved by minimizing the least-squares (LS) error $F(\boldsymbol{x}) = \frac{1}{2}\sum_{i=1}^{N} \|o_i - x_i\|_2^2$. Different from centralized LS, here the observation o_i is available only to node i, and all nodes collaborate to obtain the final solution. In D-CADMM each node can only talk to its neighbors, while in H-CADMM nodes can potentially communicate with the LFC and their neighbors.

We test D-CADMM and H-CADMM solvers with various parameter settings. We assess convergence using the relative accuracy metric defined as $\|x^k - x^\star\|_2/\|x^\star\|_2$, and report the number of iterations as well as the communication cost involved in reaching a target level of performance. The communication cost measures how many times local and global decision variables are transferred across the network. Originally, we set ρ according to Theorem 2, but this choice did not work well our tests. For this reason, we tuned it manually to reach the best possible performance.

2.6.2 Acceleration of dedicated FCs

In this test, we compare the performance of H-CADMM with dedicated FCs against that of D-CADMM. In particular, we choose only one dedicated LFC connected to 20% and 50% of the nodes drawn randomly from (i) a lollipop graph; (ii) a caveman graph; and (iii) two Erdős Rényi random graphs. All the graphs have $N = 50$ nodes. Specifically, in the lollipop graph, 50% of nodes comprise a clique and the rest form a line graph attached to this clique. The caveman graph consists of a cycle formed by 10 small cliques, each forming a complete graph of 5 nodes. The Erdős Rényi graphs are randomly generated with edge probability $r = 0.05$ and $r = 0.1$, respectively.

Figure 2.5 compares the performance of H-CADMM with one dedicated FC connected to 20% and 50% of the nodes against D-CADMM, in terms of number of iterations needed to achieve certain accuracy, as well as, communication cost. Lines with the same color markers

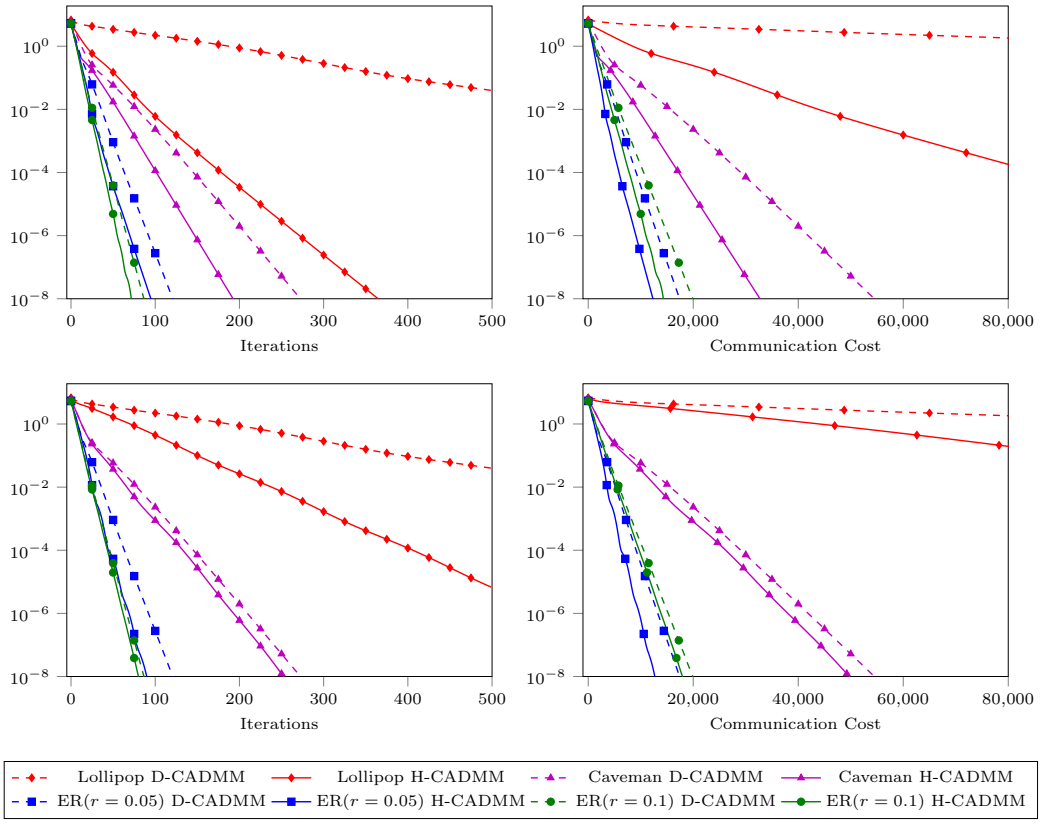


Figure 2.5: Performance comparison of D-CADMM and H-CADMM in terms of iteration number as well as communication cost. H-CADMM is configured with one dedicated FC connecting 50% of the nodes (top row figures) and 20% of the nodes respectively (bottom row figures).

denote the same graph; dashed lines correspond to D-CADMM, and solid lines to H-CADMM. From these two figures, one can draw several interesting observations:

- H-CADMM with mixed updates works well in practice. Solutions are obtained in fewer iterations, and at lower communication cost.
- The tests verify the linear convergence properties of both D-CADMM and H-CADMM, as can be seen in all four graphs.
- The performance gap between dashed lines and solid lines of the same marker confirms the acceleration ability of H-CADMM. The gap is larger for "badly-connected" graphs, such as the lollipop graph, and relatively small for "well-connected" graphs, such as

the Erdős Rényi graphs. In fact, this observation holds even when comparing the Erdős Rényi graphs. Indeed, for the ER($r = 0.05$) graph which is not as well connected as the ER($r = 0.1$), the performance gap is smaller.

- Figure 2.5 suggests that the more nodes are connected to the FC, the larger the acceleration gains for H-CADMM, which is intuitively reasonable since extra connections pay off. In view of this connections-versus-acceleration trade-off, H-CADMM can reach desirable sweet spots between performance gains and deployment cost.

2.6.3 In-network acceleration

In this test, we demonstrate the performance gain effected by in-network acceleration. By creating virtual FCs among nodes, this technique does not require dedicated FCs and new links. As detailed in Section 2.5, in-network accelerated H-CADMM exchanges information along existing edges, essentially leading to a communication cost that follows the same pattern with iteration complexity. Therefore, in this experiment, we report both metrics in one figure.

We first apply H-CADMM with in-network acceleration to several fixed-topology graphs, namely the line graph, the cycle graph, and the star graph, and we report the results in Figure 2.6. Then, we carry out the same tests on the lollipop graph, the caveman graph and the two Erdos-Renyi graphs with parameters $r = 0.05$ and $r = 0.10$, and we present the results in Figure 2.7. In each test, we select the hosts using Algorithm 2.

All the results illustrate that H-CADMM with in-network acceleration offers a significant boost in convergence rate over D-CADMM, especially for graphs with relatively large diameters (or graphs that are not well-connected), such as the line graph, the cycle graph or the lollipop graph. On the other hand, the performance gain is minimal for the star graph, whose diameter is 2 regardless of the number of nodes, as well as the Erdős Rényi random graph with high edge probability (see Figure 2.7). Note that these performance gains over D-CADMM are achieved without paying a substantial computational cost (just one averaging step), which speaks for the practical merits of H-CADMM.

2.6.4 Trade-off between FCs and performance gain

Finally, we explore the trade-off between the number of LFCs and the corresponding convergence rate. We perform tests on several graphs with different properties, and using only H-CADMM with in-network acceleration. In this test, we measure performance by the number of iterations needed to achieve a target accuracy of 10^{-8}, given a varying number of LFCs ranging from 1 to 25.

Figure 2.8 depicts the results. In general, as the number of LFCs increases, the number of iterations decreases. Besides this general trend, one can also make the following observations.

- For Erd˝os Rényi graphs with a relatively high edge probability, there is a small initial gain arising from the introduction of virtual LFCs, which diminishes fast as their number increases. This is not surprising since such graphs are "well connected," and therefore, adding more virtual LFCs does not help much. On the other hand, for "badly connected" graphs, such as the line graph or the lollipop graph, there is a significant convergence boost as the number of LFCs increases.

- For the line and the cycle graph, a significant change in convergence rate happens only after an initial threshold has been surpassed (in our case 5-6 LFCs). For the lollipop graph, there seems to exist a cut-off point above which adding more nodes does not lead to significant change in convergence rate.

2.7 Chapter summary

This chapter introduces the novel H-CADMM algorithm that generalizes the centralized and the decentralized CADMM, while also accelerating D-CADMM with modified updates.

We establish linear convergence of H-CADMM, and we also conduct a comprehensive set of numerical tests that validate our analytical findings and demonstrate the effectiveness of the proposed approach in practice.

A very promising direction we are currently pursuing involves the development of techniques leveraging the intrinsic hierarchical organization that is commonly found in distributed system architectures (see e.g., [46, 86]) as well as real-world large-scale network topologies (see e.g. [25, 85]).

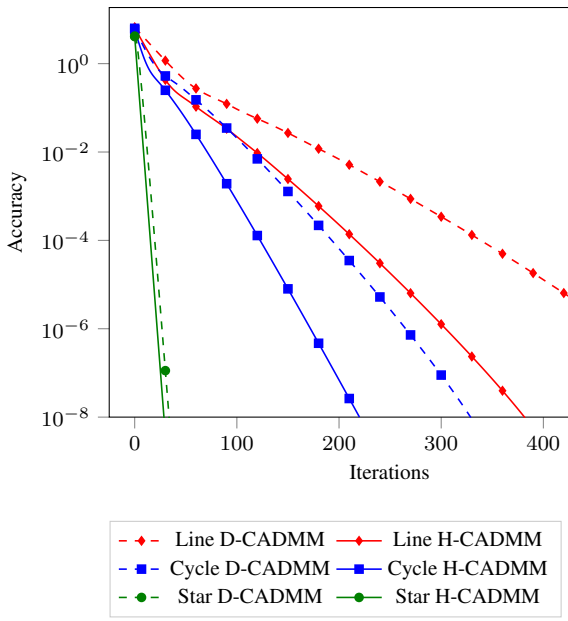


Figure 2.6: Effects of in-network acceleration on the line graph, the cycle graph, and the star graph.

2.8 Proofs of lemmas and theorems

2.8.1 Algorithm 1 for $l > 2$

For $l \geq 2$, we have $\boldsymbol{x} \in \mathbb{R}^{Nl}$, $\boldsymbol{z} \in \mathbb{R}^{Ml}$, $\boldsymbol{\lambda} \in \mathbb{R}^{Tl}$, and $\boldsymbol{y} \in \mathbb{R}^{Nl}$. Let $\tilde{A} \in \mathbb{R}^{Tl \times Nl}$ and $\tilde{B} \in \mathbb{R}^{Tl \times Ml}$ denote the coefficient matrices for $l \geq 2$, obtained by replacing 1's of $\boldsymbol{A}$ and $\boldsymbol{B}$ with the identity matrix $\boldsymbol{I}_l$ and 0's with all-zero matrix $\boldsymbol{0}_{l \times l}$. Consequently, node degree, edge degree, and incidence matrices are $\tilde{D} = \tilde{A}^\top \tilde{A} \in \mathbb{R}^{Nl \times Nl}$, $\tilde{E} = \tilde{B}^\top \tilde{B} \in \mathbb{R}^{Ml \times Ml}$, and $\tilde{C} = \tilde{A}^\top \tilde{B} \in \mathbb{R}^{Nl \times Ml}$. Meanwhile, H-CADMM boils down to

$$\boldsymbol{x}^{k+1} = (\nabla f + \rho \tilde{D} I)^{-1}(c\tilde{C}\boldsymbol{z}^k - \boldsymbol{y}^k) \tag{2.44a}$$

$$\boldsymbol{z}^{k+1} = \tilde{E}^{-1}\tilde{C}^\top \boldsymbol{x}^{k+1} \tag{2.44b}$$

$$\boldsymbol{y}^{k+1} = \boldsymbol{y}^k + \rho(\tilde{D}\boldsymbol{x}^{k+1} - \tilde{C}\boldsymbol{z}^{k+1}). \tag{2.44c}$$

Relative to (2.18)–(2.20), the computational complexity of (2.44) is clearly higher. To see this, consider the per-step complexity for $l = 1$. Given that $\boldsymbol{E}$ is a diagonal matrix, naive matrix

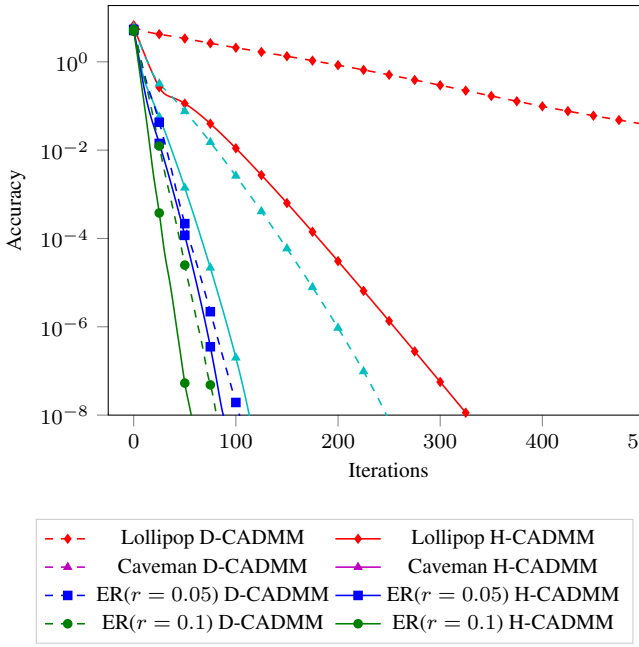


Figure 2.7: Effects of in-network acceleration on the lollipop graph, the caveman graph, and two Erdős Rényi random graphs.

vector multiplication incurs complexity $\mathcal{O}(MN)$, which can be reduced to $\mathcal{O}(ME_{\max})$ by exploiting the sparsity of $\boldsymbol{C}$, where $E_{\max}$ is the largest edge degree. When $l \geq 2$ however, per-step complexity grows to $\mathcal{O}(l^2 ME_{\max})$ which – being quadratic in l – would make it difficult for the method to handle high-dimensional data. Thankfully, a compact form of (2.44) made possible by Proposition 3 reduces the complexity from quadratic to linear in l.

Proposition 3. *Let* $\boldsymbol{X} \in \mathbb{R}^{N\times l}$ *denote the matrix formed with* i*-th row* $\boldsymbol{x}_i^\top$ *and likewise for* $\boldsymbol{Z} \in \mathbb{R}^{M\times l}$ *and* $\boldsymbol{Y} \in \mathbb{R}^{N\times l}$*. Then,* (2.44) *is equivalent to*

$$\boldsymbol{X}^{k+1} = (\nabla f + \rho \boldsymbol{D} I)^{-1}(\rho \boldsymbol{C}\boldsymbol{Z}^k - \boldsymbol{Y}^k) \tag{2.45a}$$

$$\boldsymbol{Z}^{k+1} = \boldsymbol{E}^{-1}\boldsymbol{C}^\top \boldsymbol{X}^{k+1} \tag{2.45b}$$

$$\boldsymbol{Y}^{k+1} = \boldsymbol{Y}^k + \rho(\boldsymbol{D}\boldsymbol{X}^{k+1} - \boldsymbol{C}\boldsymbol{Z}^{k+1}). \tag{2.45c}$$

Proof. The block structure suggests the following compact representation using Kronecker products

$$\tilde{C} = \boldsymbol{C} \otimes \boldsymbol{I}_d, \quad \tilde{D} = \boldsymbol{D} \otimes \boldsymbol{I}_d, \quad \tilde{E} = \boldsymbol{E} \otimes \boldsymbol{I}_d.$$

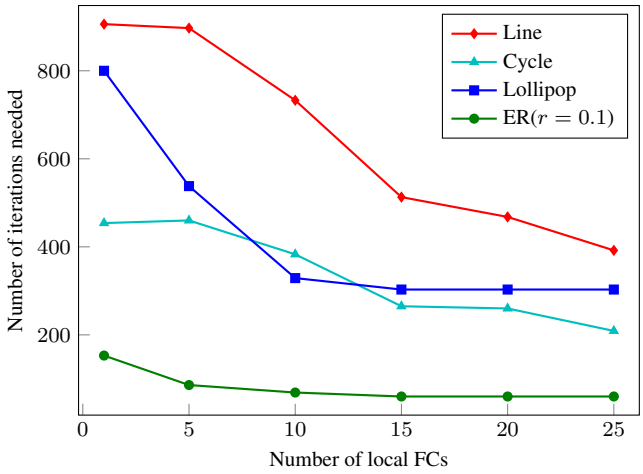


Figure 2.8: The impact of adding LFCs in H-CADMM with in-network acceleration. The performance is measured in terms of number of iterations needed to achieve certain accuracy.

Exploiting properties of Kronecker products [55, §2.8], matrices $\boldsymbol{P}$, $\boldsymbol{Q}$, $\boldsymbol{R}$, and $\boldsymbol{S}$ with compatible dimensions, satisfy

$$(\boldsymbol{P} \otimes \boldsymbol{Q}^\top)\boldsymbol{x} = \text{vec}(\boldsymbol{P}\boldsymbol{X}\boldsymbol{Q})$$

where $\boldsymbol{x}$ is obtained by concatenating all rows of $\boldsymbol{X}$ that we denote as $\boldsymbol{x} = \text{vec}(\boldsymbol{X}^\top)$. Thus, by setting $\boldsymbol{P} = \boldsymbol{C}$ and $\boldsymbol{Q} = \boldsymbol{I}_d$, one arrives at

$$\begin{aligned}\tilde{\boldsymbol{C}}\boldsymbol{z} &= (\boldsymbol{C} \otimes \boldsymbol{I}_d)\boldsymbol{z} = \text{vec}(\boldsymbol{C}\boldsymbol{Z}\boldsymbol{I}_d) = \text{vec}(\boldsymbol{C}\boldsymbol{Z}) \\ \tilde{\boldsymbol{D}}\boldsymbol{x} &= (\boldsymbol{D} \otimes \boldsymbol{I}_d)\boldsymbol{x} = \text{vec}(\boldsymbol{D}\boldsymbol{X}\boldsymbol{I}_d) = \text{vec}(\boldsymbol{D}\boldsymbol{X})\end{aligned}$$

from which one readily obtains (2.45a) and (2.45c). To see the equivalence of (2.44b) and (2.45b), we use another property of Kronecker products, namely

$$(\boldsymbol{P} \otimes \boldsymbol{Q})(\boldsymbol{R} \otimes \boldsymbol{S}) = (\boldsymbol{P}\boldsymbol{R}) \otimes (\boldsymbol{Q}\boldsymbol{S}).$$

Since $\tilde{E}$ is block diagonal, so is $\tilde{E}^{-1} = \boldsymbol{E}^{-1} \otimes \boldsymbol{I}_d$. Therefore, (2.44b) is equivalent to

$$\begin{aligned}\tilde{\boldsymbol{E}}^{-1}\tilde{\boldsymbol{C}}^\top\boldsymbol{x} &= (\boldsymbol{E}^{-1} \otimes \boldsymbol{I}_d)(\boldsymbol{C}^\top \otimes \boldsymbol{I}_d)\text{vec}(\boldsymbol{X}) \\ &= ((\boldsymbol{E}^{-1}\boldsymbol{C}^\top) \otimes \boldsymbol{I}_d)\text{vec}(X) = \text{vec}(\boldsymbol{E}^{-1}\boldsymbol{C}^\top\boldsymbol{X})\end{aligned} \tag{2.46}$$

which concludes the proof. □

Proposition 3 establishes that H-CADMM can run using much smaller matrices, effectively reducing its computational complexity. To see this, consider the per-step complexity of (2.45). The difference with (2.44) is dominated by $\boldsymbol{C}^\top \boldsymbol{X}$, which leads to complexity $\mathcal{O}(lMN)$. This can be further reduced to $\mathcal{O}(lME_{\text{max}})$ by exploiting the sparsity of $\boldsymbol{C}$, thus improving the complexity of (2.44) by a factor of l.

2.8.2 Proof of Lemma 1

Let $\boldsymbol{a}_i$ denote the i-th column of $\boldsymbol{A}$, and $\boldsymbol{b}_j$ the j-th column of $\boldsymbol{B}$. By construction, the i-th column of $\boldsymbol{A}$ indicates in which constraint x_i is present; hence, $\boldsymbol{a}_i^\top \boldsymbol{a}_i = d_i$ equals the degree of node i. Similarly, it follows that $\boldsymbol{b}_j^\top \boldsymbol{b}_j = e_j$, and

$$\begin{aligned} \boldsymbol{a}_i^\top \boldsymbol{a}_j &= 0, \quad \forall i \neq j \\ \boldsymbol{b}_i^\top \boldsymbol{b}_j &= 0, \quad \forall i \neq j \end{aligned}$$

from which we obtain $\boldsymbol{A}^\top \boldsymbol{A} = \boldsymbol{D}$ and $\boldsymbol{B}^\top \boldsymbol{B} = \boldsymbol{E}$.

Consider now the dot product $\boldsymbol{a}_i^\top \boldsymbol{b}_j$. When the t-th constraint reads $x_i = z_j$, it holds by construction that $(\boldsymbol{a}_i)_t = 1$, and $(\boldsymbol{b}_j)_t = 1$. Therefore, if node i and edge j are incident, we have $\boldsymbol{a}_i^\top \boldsymbol{b}_j = 1$; otherwise, $\boldsymbol{a}_i^\top \boldsymbol{b}_j = 0$. Thus, the incidence matrix definition implies that $\boldsymbol{A}^\top \boldsymbol{B} = \boldsymbol{C}$.

2.8.3 Proof of Lemma 3

Let e_{max} (e_{min}) denote the maximum (minimum) degree of all edges, and c_{ij} the number of common edges between nodes i and j. Clearly, we have $d_i = \sum_{j=1}^{N} c_{ij}$, with $c_{ii} = 0$; and since $\boldsymbol{E} \preceq e_{\text{max}} \boldsymbol{I}$, we obtain

$$\boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top \succeq \frac{1}{e_{\text{max}}} \boldsymbol{C}\boldsymbol{C}^\top \succeq \boldsymbol{0}$$

where the last inequality holds because $\boldsymbol{C}$ has linearly independent columns, and hence $\boldsymbol{C}\boldsymbol{C}^\top$ is PSD. The latter implies that $\boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top$ is PSD too.

The definition of $\boldsymbol{C}$ leads to

$$\boldsymbol{C}\boldsymbol{C}^\top = \begin{bmatrix} d_1 & c_{12} & \dots & c_{1N} \\ c_{21} & d_2 & \dots & c_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ c_{N1} & c_{N2} & \dots & d_N \end{bmatrix}.$$

And since $\boldsymbol{E} \succeq e_{\min}\boldsymbol{I}$, it holds that

$$\begin{aligned} \boldsymbol{D} - \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top &\succeq \boldsymbol{D} - \frac{1}{e_{\min}}\boldsymbol{C}\boldsymbol{C}^\top \\ &\succeq \boldsymbol{D} - \frac{1}{2}\boldsymbol{C}\boldsymbol{C}^\top \\ &= \frac{1}{2}\begin{bmatrix} d_1 & -c_{12} & \dots - c_{1N} & \\ -c_{21} & d_2 & \dots - c_{2N} & \\ \vdots & \vdots & \ddots & \vdots \\ -c_{N1} & -c_{N2} & \dots & d_N \end{bmatrix} \end{aligned}$$

where the last matrix is a valid Laplacian ($d_i = \sum_{j=1}^{N} c_{ij}$), which in turn implies that it is PSD.

Finally, by definition $\boldsymbol{C}\mathbf{1} = \boldsymbol{d}$, $\boldsymbol{C}^\top\mathbf{1} = \boldsymbol{e}$, where $\boldsymbol{d} := [d_1, \dots, d_N]^\top$, $\boldsymbol{e} := [e_1, \dots, e_N]^\top$; and hence we have

$$(\boldsymbol{D} - \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top)\mathbf{1} = \boldsymbol{d} - \boldsymbol{C}\mathbf{1} = \mathbf{0}.$$

2.8.4 Proof of Lemma 4

Feasibility of (2.30c) guarantees the existence of at least one solution, and therefore there exists at least one optimal solution. The uniqueness of $\boldsymbol{x}^\star$ and $\boldsymbol{z}^\star$ follows from the strong convexity of $F(\cdot)$ and the dual feasibility (2.30c).

To see the uniqueness of $\boldsymbol{y}$, we first show that if $\tilde{\boldsymbol{\lambda}}$ is optimal, then $\boldsymbol{\lambda}^\star$, the projection of $\tilde{\boldsymbol{\lambda}}$ to the column space of $\boldsymbol{A}$, is also optimal. Using the orthogonality principle, we arrive at

$$\boldsymbol{A}^\top(\tilde{\boldsymbol{\lambda}} - \boldsymbol{\lambda}^\star) = \mathbf{0}$$

which implies that $\boldsymbol{\lambda}^\star$ also satisfies (2.30a). Thus, projection of any solution $\tilde{\boldsymbol{\lambda}}$ to the column space of $\boldsymbol{A}$ is also an optimal solution.

If there are two optimal solutions, $\boldsymbol{\lambda}_1 \neq \boldsymbol{\lambda}_2$ in the column space of $\boldsymbol{A}$, we have

$$\boldsymbol{A}^\top \boldsymbol{\lambda}_1 = \boldsymbol{A}^\top \boldsymbol{\lambda}_2. \tag{2.47}$$

Furthermore, there exist $\boldsymbol{x}_1 \neq \boldsymbol{x}_2$ such that $\boldsymbol{\lambda}_1 = \boldsymbol{A}\boldsymbol{x}_1$, and $\boldsymbol{\lambda}_2 = \boldsymbol{A}\boldsymbol{x}_2$. Subtracting $\boldsymbol{A}^\top \boldsymbol{\lambda}_2$ from $\boldsymbol{A}^\top \boldsymbol{\lambda}_1$ yields

$$\boldsymbol{A}^\top \boldsymbol{\lambda}_1 - \boldsymbol{A}^\top \boldsymbol{\lambda}_2 = \boldsymbol{A}^\top \boldsymbol{A}(\boldsymbol{x}_1 - \boldsymbol{x}_2) = \boldsymbol{0} \tag{2.48}$$

from which we find that $\boldsymbol{x}_1 = \boldsymbol{x}_2$, which is a contradiction. Therefore, $\boldsymbol{\lambda}_1 = \boldsymbol{\lambda}_2$, and thus $\boldsymbol{y} = \boldsymbol{A}^\top \boldsymbol{\lambda}$ is also unique.

2.8.5 Proof of Lemma 5

To simplify the notation, let $\mathbf{S} := \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top$. Using Lemma 2, we arrive at

$$\boldsymbol{D}\boldsymbol{x}^{k+1} = -\frac{1}{\rho}\nabla F(\boldsymbol{x}^{k+1}) + \mathbf{S}\boldsymbol{x}^k - (\boldsymbol{D} - \mathbf{S})\sum_{t=0}^{k} \boldsymbol{x}^t.$$

Subtracting $(\boldsymbol{D} - \mathbf{S})\boldsymbol{x}^{k+1}$ from both sides yields

$$\mathbf{S}(\boldsymbol{x}^{k+1} - \boldsymbol{x}^k) = -\frac{1}{\rho}\nabla F(\boldsymbol{x}^{k+1}) - (\boldsymbol{D} - \mathbf{S})\sum_{t=0}^{k+1} \boldsymbol{x}^t.$$

Noticing that $\boldsymbol{Q} = (\boldsymbol{D} - \mathbf{S})^{1/2}$ and $\boldsymbol{r}^k = \sum_{t=0}^{k} \boldsymbol{Q}\boldsymbol{x}^t$, we obtain

$$\mathbf{S}(\boldsymbol{x}^{k+1} - \boldsymbol{x}^k) = -\frac{1}{\rho}\nabla F(\boldsymbol{x}^{k+1}) - \boldsymbol{Q}\boldsymbol{r}^{k+1}. \tag{2.49}$$

The proof of Lemma 2 shows that if $\boldsymbol{y}^0 = \boldsymbol{0}$, then $\boldsymbol{y}^k = \rho\boldsymbol{Q}\boldsymbol{r}^k$, which leads to $\boldsymbol{y}^\star = \rho\boldsymbol{Q}\boldsymbol{r}^\star$ as $k \to \infty$. Using (2.30), we arrive at

$$\nabla F(\boldsymbol{x}^\star) = -\rho\boldsymbol{Q}\boldsymbol{r}^\star.$$

Combining this with (2.49) completes the proof.

2.8.6 Proof of Theorem 1

It suffices to prove that

$$\|\boldsymbol{q}^{k+1}-\boldsymbol{q}^{\star}\|_{\boldsymbol{G}}^2 \leq \frac{1}{1+\delta}\|\boldsymbol{q}^{k}-\boldsymbol{q}^{\star}\|_{\boldsymbol{G}}^2. \tag{2.50}$$

Lemma 5 and the strong convexity of $F(\cdot)$ lead to

$$\begin{aligned}
\frac{2}{\rho}\sigma\|\boldsymbol{x}^{k+1}-\boldsymbol{x}^{\star}\|_2^2 &\leq \frac{2}{\rho}(\boldsymbol{x}^{k+1}-\boldsymbol{x}^{\star})^{\top}(\nabla F(\boldsymbol{x}^{k+1})-\nabla F(\boldsymbol{x}^{\star})) \\
&= -2(\boldsymbol{r}^{k+1}-\boldsymbol{r}^{\star})^{\top}\boldsymbol{Q}(\boldsymbol{r}^{k+1}-\boldsymbol{r}^{\star}) \\
&\quad -2(\boldsymbol{x}^{k+1}-\boldsymbol{x}^{k})^{\top}\boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^{\top}(\boldsymbol{x}^{k+1}-\boldsymbol{x}^{\star}) \\
&= 2(\boldsymbol{q}^{k+1}-\boldsymbol{q}^{k})^{\top}\boldsymbol{G}(\boldsymbol{r}^{k+1}-\boldsymbol{r}^{\star}) \\
&= \|\boldsymbol{q}^{k}-\boldsymbol{q}^{\star}\|_{\boldsymbol{G}}^2-\|\boldsymbol{q}^{k+1}-\boldsymbol{q}^{\star}\|_{\boldsymbol{G}}^2-\|\boldsymbol{q}^{k}-\boldsymbol{q}^{k+1}\|_{\boldsymbol{G}}^2.
\end{aligned}$$

Similarly, Lemma 5 and strong convexity imply that

$$\frac{2}{\rho}\frac{1}{L}\|\nabla F(\boldsymbol{x}^{k+1})-\nabla F(\boldsymbol{x}^{\star})\|_2^2 \leq \|\boldsymbol{q}^{k}-\boldsymbol{q}^{\star}\|_{\boldsymbol{G}}^2-\|\boldsymbol{q}^{k+1}-\boldsymbol{q}^{\star}\|_{\boldsymbol{G}}^2-\|\boldsymbol{q}^{k}-\boldsymbol{q}^{k+1}\|_{\boldsymbol{G}}^2.$$

For any $\beta \in (0,1)$, we have

$$\begin{aligned}
\beta\frac{2}{\rho}\sigma\|\boldsymbol{x}^{k+1}-\boldsymbol{x}^{\star}\|_2^2+(1-\beta)\frac{2}{\rho}\frac{1}{L}\|\nabla F(\boldsymbol{x}^{k+1})-\nabla F(\boldsymbol{x}^{\star})\|_2^2 & \\
\leq \|\boldsymbol{q}^{k}-\boldsymbol{q}^{\star}\|_{\boldsymbol{G}}^2-\|\boldsymbol{q}^{k+1}-\boldsymbol{q}^{\star}\|_{\boldsymbol{G}}^2-\|\boldsymbol{q}^{k}-\boldsymbol{q}^{k+1}\|_{\boldsymbol{G}}^2. &
\end{aligned}$$

To prove (2.50), it suffices to show that

$$\begin{aligned}
\|\boldsymbol{q}^{k}-\boldsymbol{q}^{k+1}\|_{\boldsymbol{G}}^2+\beta\frac{2}{\rho}\sigma\|\boldsymbol{x}^{k+1}-\boldsymbol{x}^{\star}\|_2^2+(1-\beta)\frac{2}{\rho}\frac{1}{L}\|\nabla F(\boldsymbol{x}^{k+1}-\nabla F(\boldsymbol{x}^{\star}))\|_2^2 & \\
\geq \delta\|\boldsymbol{q}^{k+1}-\boldsymbol{q}^{\star}\|_{\boldsymbol{G}}^2 & \quad (2.51)
\end{aligned}$$

which is equivalent to

$$\begin{aligned}
(1-\beta)\frac{2}{\rho}\frac{1}{L}\|\nabla F(\boldsymbol{x}^{k+1})-\nabla F(\boldsymbol{x}^{\star})\|_2^2+\|\boldsymbol{q}^{k}-\boldsymbol{q}^{k+1}\|_{\boldsymbol{G}}^2+\|\boldsymbol{x}^{k+1}-\boldsymbol{x}^{\star}\|_{\boldsymbol{M}}^2 & \\
\geq \delta\|\boldsymbol{r}^{k+1}-\boldsymbol{r}^{\star}\|_2^2 & \quad (2.52)
\end{aligned}$$

where $\boldsymbol{M} := \frac{2\sigma\beta}{\rho}\boldsymbol{I} - \delta\boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top$. Observing the left hand side of (2.52), it suffices to show that

$$\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^\star\|_{\boldsymbol{M}}^2 + (1-\beta)\frac{2}{\rho L}\|\nabla F(\boldsymbol{x}^{k+1}) - \nabla F(\boldsymbol{x}^\star)\|_2^2 \geq \delta\|\boldsymbol{r}^{k+1} - \boldsymbol{r}^\star\|_2^2. \tag{2.53}$$

Since $\boldsymbol{r}^{k+1}$ and $\boldsymbol{r}^\star$ are orthogonal to $\mathbf{1}$ and the null space of $\boldsymbol{Q}$ is span$\{\mathbf{1}\}$, both vectors belong to the column space of $\boldsymbol{Q}$. Using Lemma 4, we obtain

$$\begin{aligned}
\delta\|\boldsymbol{r}^{k+1} - \boldsymbol{r}^\star\|_2^2 &\leq \frac{\delta}{\lambda_{\min}}\|\boldsymbol{Q}(\boldsymbol{r}^{k+1} - \boldsymbol{r}^\star)\|_2^2 \\
&\leq \frac{\delta}{\lambda_{\min}}\|\boldsymbol{M}(\boldsymbol{x}^{k+1} - \boldsymbol{x}^\star) - \frac{1}{\rho}(\nabla F(\boldsymbol{x}^{k+1}) - \nabla F(\boldsymbol{x}^\star))\|_2^2 \\
&\leq \frac{2\delta\lambda_{\max}}{\lambda_{\min}}\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^\star\|_{\boldsymbol{M}}^2 + \frac{2\delta}{\rho\lambda_{\min}}\|\nabla F(\boldsymbol{x}^{k+1}) - \nabla F(\boldsymbol{x}^\star)\|_2^2.
\end{aligned} \tag{2.54}$$

Comparing (2.54) with (2.53) suggests that it is sufficient to have

$$\delta \leq \min\left\{\frac{2\beta\sigma}{\rho(\lambda_{\max} + \frac{2\lambda_{\max}}{\lambda_{\min}})}, \frac{(1-\beta)\rho\lambda_{\min}}{L}\right\}. \tag{2.55}$$

Chapter 3

Graph-aware Weighted Hybrid ADMM for Fast Decentralized Optimization

3.1 Introduction

Unprecedented amount of data and computation power demand of large-scale problems call for distributed processing. As a key ingredient, distributed optimization has attracted increasing attention in many areas, including machine learning [60], Internet-of-Things (IoT) [47], distributed detection and estimation [37, 100], to name a few. In the setting of distributed optimization, a group of networked computing nodes, each holding its own privately available data, work together to minimize some objective function. Every node makes its decision by minimizing its local cost function as well as collaborating with others.

Among various solvers for decentralized optimization problem, alternating method of multipliers (ADMM) stands out because of its simplicity, fast convergence and decomposability [5, 9]. Depending the existence of a central coordinator, distributed optimization problem can be solved in centralized [9] or decentralized manner [37, 103], which we abbreviate as CADMM and DAMM. The present work focuses on the decentralized case. In a nutshell, each iteration of DADMM comprises: (i) an update step where each node decreases its own cost based on information of its neighbors; and (ii) a communication step where nodes exchange information

with their single-hop neighbors. DADMM treats each node equally regardless of how many neighbors it is connected. This extra information could be used to characterize the importance of a node and possibly leads to some smart ways of solving the same problem. This chapter propose the weighted hybrid ADMM (WHADMM) that relies on the same two-step procedure but is capable of exploiting local topology information.

Related works. There are many methods for solving distributed optimization problem, including (sub)gradient methods[82], Nesterov accelerated (sub)gradient [56], dual averaging [21], gossip [20], ADMM [103, 37] and accelerated ADMM [43]. Among these methods, ADMM features the ability to decompose complex cost functions to simple sub-problems that can be easily handled, and enjoys fast convergence to moderately accurate solutions [9]. When cost functions are strongly convex and Lipschitz continuous ADMM converges linearly to optimal solutions [19, 51, 103]. However, most algorithms consider only unweighted graphs and treat all edges equally. Weighted decentralized ADMM (WDADMM) [65] takes edge weights into consideration and was shown to outperform its unweighted counterparts.

Existing ADMM-based algorithms achieve consensus by exchanging information with single-hop neighbors while being unaware of local topology. The hybrid ADMM (HADMM) [71] carefully considers local neighborhood structure to enable faster information flow so as to achieve better convergence. However, HADMM works only for undirected graphs.

Contributions. The present chapter introduces WHADMM algorithm, which can cope with weighted graphs, and also is capable of exploiting topology information. Moreover, for certain types of network, WHADMM can distinguish important edges from unimportant ones, and emphasize more on critical edges by assigning larger weights. Unlike WDADMM, which works best for graphs with clusters [65], WHADMM shows improvement over DADMM on almost any kind of graphs.

3.2 Background and problem statement

The distributed optimization problems aims to minimize some cost function $f(\boldsymbol{x}) = \sum_i f_i(\boldsymbol{x})$ by finding a common optimal decision variable $\boldsymbol{x} \in \mathbb{R}^p$, where the local cost function f_i is privately available to node i only and contains node-private data [37]. The decision variable $\boldsymbol{x}$ is shared by all nodes, making problem coupled and preventing parallel processing. A common strategy is to create local copies at each node to decouple the problem. The same optimal

solution can be obtained by enforcing consensus over the whole network to ensure all local copies agree with global decision variable. With this technique, the problem can be formulated as

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \sum_i f_i(\boldsymbol{x}_i) \\ \text{s. to} \quad & \boldsymbol{x}_1 = \boldsymbol{x}_2 = \boldsymbol{x}_3 = \dots \end{aligned} \tag{3.1}$$

where N is the number of nodes and $\boldsymbol{x}_i$ is the local copy of global decision variable at node i. To reach consensus, each node needs to communicate with others. Such communication may be constrained by many factors, such as location, physical connectivity, or privacy concerns.

The communication constraints of a network can be conveniently specified by a graph, denoted by a tuple $\mathscr{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, 2, \dots, N\}$ is the vertex set corresponding to N networked nodes, and $\mathcal{E} := \{(i, j) | i, j \in \mathcal{V}\}$ is the edge set indicating the connectivity of between nodes. Nodes i and j can communicate if $(i, j) \in \mathcal{E}$. We only consider undirected graphs with no self-loops.

The communication constraints of HADMM are the same as those of DADMM, but the actual exchange of information, represented by a hypergraph [12], is different, in order to take advantage of topology information. A hypergraph is also a tuple $\mathscr{H} = (\mathcal{E}, \mathcal{V})$, where $\mathcal{V} := \{1, 2, \dots, N\}$ is the vertex set and $\mathcal{E} := \{\mathcal{E}_i | \mathcal{E}_i \subseteq V\}$ is the edge set, each edge $\mathcal{E}_i$ being a subset of $\mathcal{V}$. A hyperedge may consist of multiple nodes. Nodes in the same hyperedge are neighbors of each other and may exchange information. The hypergraph degrades to a simple graph when every hyperedge comprises exactly two nodes, i.e., $|\mathcal{E}_i| = 2$. Therefore, hypergraphs are a generalization of simple graphs and provide a unifying way to describe communication constraints.

Let $\mathscr{H}$ be a hypergraph with N nodes and M hyperedges. The incidence matrix $\boldsymbol{C} \in \mathbb{R}^{N \times M}$ of $\mathscr{H}$ is defined such that $c_{ij} = 1$ if node i and edge j are connected, and $c_{ij} = 0$ otherwise. The degree of node i is the total number of incident edges, i.e., $d_i = \sum_j C_{ij}$; the degree of edge j is the total number of incident nodes, i.e., $e_j = \sum_i c_{ij}$. Let $\boldsymbol{D} = \operatorname{diag}(\boldsymbol{d})$ and $\boldsymbol{E} = \operatorname{diag}(\boldsymbol{e})$ be diagonal matrices, where $\boldsymbol{d}$ and $\boldsymbol{e}$ are vectors collecting d_i and e_j.

Problem statement. Given N networked nodes, whose connectivity is described by a hypergraph $\mathscr{H}$, the decentralized optimization problem can be reformulated as

$$\begin{aligned} \underset{\{\boldsymbol{x}_i\},\{\boldsymbol{z}_j\}}{\text{minimize}} \quad & \sum_i f_i(\boldsymbol{x}_i) \\ \text{subject to} \quad & \boldsymbol{x}_i = \boldsymbol{z}_j \quad i \in \mathcal{E}_j \end{aligned} \tag{3.2}$$

where $\boldsymbol{z}_j \in \mathbb{R}^p$ is the auxiliary variable associated with hyperedge j, playing similar a role to a *local fusion center* [71]. As long as the hypergraph is connected, problem (3.2) and (3.1) are equivalent, sharing the same solutions.

3.3 Weighted hybrid ADMM

The decentralized optimization problem (3.2) can be solved efficiently using HADMM. By creating hypergraphs properly, HADMM can take advantage of local connectivity, achieve faster convergence and save communication cost [71].

Example. Consider the graph in Fig. 3.1. There are three hyperedges, namely $\mathcal{E}_1 := \{1 \dots 5\}$, $\mathcal{E}_2 := \{6 \dots 10\}$, and $\mathcal{E}_3 := \{5, 10, 11\}$, shown as gray dashed circles. Iterative steps of ADMM solving (3.2) involve information exchange between $\boldsymbol{x}_2$ and $\boldsymbol{z}_1$, which can be implemented via communication between nodes 2 and 1, upon realizing $\boldsymbol{z}_1$ can be viewed as a *virtual fusion center* [71] residing in node 1. Similar tricks can be done for $\mathcal{E}_2$ and $\mathcal{E}_3$. By creating *virtual fusion centers*, HADMM can employ topology information using the same communication network as DADMM.

HADMM works only for unweighted graphs, treating every edge equally. WHADMM, on the other hand, takes edge importance into consideration based on different edge weights. Even when edge weights are not readily available, WHADMM can identify important edges and assign larger weights.

Denote $\mathcal{D} \in \mathbb{R}_{++}^{N\times N}$ the set of diagonal positive definite matrices, $\mathcal{C} \in \mathbb{R}_{+}^{N\times N}$ the set of matrices whose (i,j)-th elements are nonzero if and only if $c_{ij} = 1$. Let $\boldsymbol{X} = [\boldsymbol{x}_1^\top; \dots; \boldsymbol{x}_N^\top] \in \mathbb{R}^{N\times p}$, $\boldsymbol{Y} = [\boldsymbol{y}_1^\top; \dots; \boldsymbol{y}_N^\top] \in \mathbb{R}^{N\times p}$ where $\boldsymbol{y}_i \in \mathbb{R}^p$ is the concatenation of dual variables

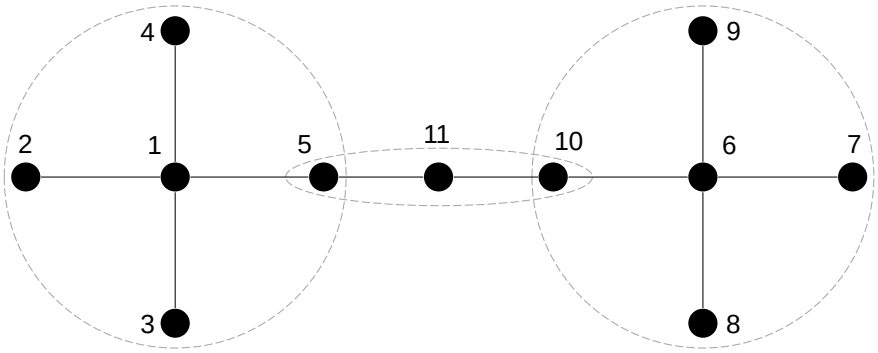


Figure 3.1: A graph consisting of two clusters. Solid black circles represent nodes, solid lines indicate connectivity between nodes and dashed circles identify hyperedges.

$\boldsymbol{y}_i \in \mathbb{R}^p$. The WHADMM algorithm can be expressed by iterative updates

$$\begin{aligned} \boldsymbol{X}^{k+1} &= \arg\min_{\boldsymbol{X}} f(\boldsymbol{X}) + \frac{\rho}{2}\langle \boldsymbol{X}, \boldsymbol{D}_w \boldsymbol{X}\rangle + \langle \boldsymbol{X}, \boldsymbol{Y}^k - \rho \boldsymbol{H}\boldsymbol{H}^\top \boldsymbol{X}^k\rangle \\ \boldsymbol{Y}^{k+1} &= \boldsymbol{Y}^k + \rho(\boldsymbol{D}_w - \boldsymbol{H}\boldsymbol{H}^\top)\boldsymbol{X}^{k+1} \end{aligned} \tag{3.3}$$

where $\boldsymbol{D}_w \in \mathcal{D}$, $\boldsymbol{H} \in \mathcal{C}$, ρ is some tunable parameter and we use superscript k as iteration index.

Remark 10. The parameter ρ can be absorbed into $\boldsymbol{D}$ and $\boldsymbol{H}$. We keep it to be consistent with HADMM and DADMM.

If at least one optimal solutions of problem (3.2) exists, sequences ogenerated by WHADMM converge to one optimal solution, provided the following conditions are satisfied:

$$\boldsymbol{D}_w - \boldsymbol{H}\boldsymbol{H}^\top \succeq 0 \tag{3.4}$$

$$\boldsymbol{H}\boldsymbol{H}^\top \succeq 0 \tag{3.5}$$

$$\text{Null}(\boldsymbol{D}_w - \boldsymbol{H}\boldsymbol{H}^\top) = \text{span}\{\boldsymbol{1}\} \tag{3.6}$$

where $\text{Null}(\boldsymbol{A})$ denotes the null space of $\boldsymbol{A}$, $\{\boldsymbol{x}|\boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}\}$. These conditions ensure $\boldsymbol{D}_w - \boldsymbol{H}\boldsymbol{H}^\top$ is a valid Laplacian matrix of the hypergraph [7].

The WHADMM algorithm (3.3) is amenable to decentralized processing. Towards this end, notice both $\boldsymbol{D}$ and $\boldsymbol{H}$ have special structures that makes (3.3) separable across nodes. In

particular, when $\boldsymbol{H} = \boldsymbol{C}$, j-th row of $\boldsymbol{Z} = \boldsymbol{H}^\top \boldsymbol{X}$ can be obtained by

$$\boldsymbol{z}_j = \sum_{i=1}^{N} c_{ij} \boldsymbol{x}_i = \sum_{i=1}^{N} \mathbb{1}_{\{i \in \mathcal{E}_j\}} \boldsymbol{x}_i = \sum_{i \in \mathcal{E}_j} \boldsymbol{x}_i. \tag{3.7}$$

Consequently, if $\boldsymbol{V} = \boldsymbol{H}\boldsymbol{Z} = \boldsymbol{H}\boldsymbol{H}^\top \boldsymbol{X}$, i-th row of $\boldsymbol{V}$ can be obtained by

$$\boldsymbol{v}_i = \sum_{j=1}^{M} c_{ij} \boldsymbol{z}_j = \sum_{j=1}^{M} \mathbb{1}_{i \in \mathcal{E}_j} \sum_{k=1}^{N} \mathbb{1}_{k \in \mathcal{E}_j} \boldsymbol{x}_k = \sum_{j: i \in \mathcal{E}_j} \sum_{k \in \mathcal{E}_j} \boldsymbol{x}_k. \tag{3.8}$$

It is clear that $\boldsymbol{v}_i$ can be computed by node i through communication with neighbors. Consider the explicit WHADMM iterations at node i,

$$\begin{aligned} \boldsymbol{x}_i^{k+1} &= \arg\min_{\boldsymbol{x}_i} f_i(\boldsymbol{x}_i) + \frac{\rho}{2} d_i^w \|\boldsymbol{x}_i\|^2 + \langle \boldsymbol{x}_i, \boldsymbol{y}_i^k - \rho \boldsymbol{v}_i^k \rangle \\ \boldsymbol{y}_i^{k+1} &= \boldsymbol{y}_i^k + \rho(d_i^w \boldsymbol{x}_i^{k+1} - \boldsymbol{v}_i^{k+1}) \end{aligned} \tag{3.9}$$

where d_i^w is the i-th diagonal element of $\boldsymbol{D}_w$ and $\boldsymbol{v}_i$ is defined in (3.8). Node i only needs $\boldsymbol{v}_i$, which can be obtained by communicating with neighbors, to update $\boldsymbol{x}_i^{k+1}$ and $\boldsymbol{y}_i^{k+1}$.

3.3.1 Connections to existing algorithms

The WHADMM algorithm is closed related to several ADMM-based decentralized optimization algorithms. For example, WHADMM reduces to HADMM when $\boldsymbol{H} = \boldsymbol{C}\boldsymbol{E}^{-1/2}$. To see this, notice that in [71] HADMM are equivalent to

$$\begin{aligned} \boldsymbol{X}^{k+1} = \arg\min_{\boldsymbol{X}} f(\boldsymbol{X}) + \frac{\rho}{2} \boldsymbol{X}^\top \boldsymbol{D} \boldsymbol{X} + \\ \langle \boldsymbol{X}, \boldsymbol{Y}^k - \rho \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top \boldsymbol{X}^k \rangle \\ \boldsymbol{Y}^{k+1} = \boldsymbol{Y}^k + \rho(\boldsymbol{D} - \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top)\boldsymbol{X}^{k+1} \end{aligned} \tag{3.10}$$

where $\boldsymbol{Z}^k = \boldsymbol{E}^{-1}\boldsymbol{C}^\top \boldsymbol{X}^k$ is eliminated. Since $\boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top = \boldsymbol{H}\boldsymbol{H}^\top$, algorithm (3.3) is equivalent to (3.10). Consequently, as discussed in [71], we can recover popular distributed ADMM-based algorithms depending on the topology of underlying $\mathscr{G}$:

1. when $\mathscr{G}$ is a star graph, then $\mathscr{H}$ has only one hyperedge consisting of all nodes, i.e., $\boldsymbol{C} = \boldsymbol{1}$, and we recover centralized ADMM [5, 9];

2. when $\mathscr{G}$ is ordinary graph, then each hyperedge $\mathcal{E}_j$ contains exactly two nodes, and we recover fully decentralized ADMM [37, 103];

3. and when $\mathscr{H}$ is a ordinary hypergraph, which is the most general case, we recover HADMM [71].

3.4 Convergence analysis

3.4.1 Convergence

The ADMM algorithm has been extensively studied for decades, and its convergence is well established, see [33, 67, 43, 51, 19, 103]. Based on ADMM, WHADMM is guaranteed to converge under mild conditions, see e.g. [9, §3.2] for details.

3.4.2 Linear convergence rate

When cost functions are strongly convex and Lipschitz continuous, sequences obtained by ADMM converge linearly to some optimal solution [19, 51]. However, no explicitly rate estimate is provided in [51], and full row rank of constraint coefficient matrices is required for [19]. Linear convergence of DADMM was established in [103], where full row rank is absent. Similar results can be found in [71] since DADMM is a special case of HADMM, and the analysis of HADMM carries over to DADMM.

We now discuss the linear convergence rate of WHADMM. Define the semi-norm induced by matrix $\boldsymbol{G}$ as $\|\boldsymbol{x}\|_{\boldsymbol{G}}^2 := \boldsymbol{x}^\top \boldsymbol{G}\boldsymbol{x}$, where

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{H}\boldsymbol{H}^\top \end{bmatrix}. \tag{3.11}$$

Denote $\lambda_{\max}$ the largest eigenvalue of $\boldsymbol{H}\boldsymbol{H}^\top$, $\lambda_{\min}$ smallest nonzero eigenvalue of $\boldsymbol{D} - \boldsymbol{H}\boldsymbol{H}^\top$, $\boldsymbol{x}^\star$ the limiting point of WHADMM sequence $\{\boldsymbol{x}^k\}$. Based on the analysis of HADMM [71], we arrive at the following linear convergence result.

Theorem 3. *Suppose local cost functions f_i are σ-strongly convex and have L-Lipschitz continuous gradient, and solution of* (3.2) *exists, then for $\beta \in (0, 1)$ WHADMM* (3.3) *converges*

linearly to its optimal solution

$$\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^\star\|_G^2 \leq c\left(\frac{1}{1+\delta}\right)^k \|\boldsymbol{x}^0 - \boldsymbol{x}^\star\|_G^2 \tag{3.12}$$

where c, δ are both positive constants and δ satisfies

$$\delta \leq \min\left\{\frac{2\beta\sigma}{\rho(\lambda_{max} + \frac{2\lambda_{max}}{\lambda_{min}})}, \frac{(1-\beta)\rho\lambda_{min}}{L}\right\}. \tag{3.13}$$

Proof. Since $\boldsymbol{H}\boldsymbol{H}^\top \succeq 0$ and $\boldsymbol{D} - \boldsymbol{H}\boldsymbol{H}^\top$ is a valid Laplacian matrix, satisfying all properties of $\boldsymbol{C}\boldsymbol{E}\boldsymbol{C}^\top$ and $\boldsymbol{D} - \boldsymbol{C}\boldsymbol{E}\boldsymbol{C}^\top$, by replacing $\boldsymbol{C}\boldsymbol{E}\boldsymbol{C}^\top$ with $\boldsymbol{D} - \boldsymbol{H}\boldsymbol{H}^\top$, the proof in [71] continues to hold, which concludes the proof. □

Corollary 1. Maximizing the constant δ with respect to β and ρ, the best convergence rate bound is

$$\delta_{opt} = \frac{1}{\sqrt{\kappa_F\kappa_G(1+2\kappa_G)}} \tag{3.14}$$

where $\kappa_F := L/\sigma$ and $\kappa_G := \Lambda/\lambda$ denote the condition number of cost function and condition number of the hypergraph, respectively.

Remark 11. It's not surprising δ_{opt} depends on the cost function and graph topology. This observation suggests it is possible to improve WHADMM by optimizing graph condition number via edge weights tuning.

3.5 Optimal weights

Corollary 1 provides an explicit formula of best rate bound, thus suggesting an approach to accelerate WHADMM by optimizing the parameters κ_F and κ_G. However, for a given problem, cost functions and graph topology are prescribed, so the only possible way is to tune edge weights.

Optimizing the convergence rate boils down to maximizing δ_{opt}, which is an upper bound

of actual convergence speed. Maximizing δ_{opt} is equivalent to solving

$$\begin{aligned} \max_{\boldsymbol{D}_w, \boldsymbol{H}} \quad & \delta_{opt} \\ \text{s.t.} \quad & \boldsymbol{D}_w \in \mathcal{D}, \boldsymbol{H} \in \mathcal{C} \\ & \boldsymbol{D}_w - \boldsymbol{H}\boldsymbol{H}^\top \succeq 0, \boldsymbol{H}\boldsymbol{H}^\top \succeq 0 \\ & \text{Null}(\boldsymbol{D}_w - \boldsymbol{H}\boldsymbol{H}^\top) = \text{span}\{\mathbf{1}\}. \end{aligned} \tag{3.15}$$

Problem (3.15) is difficult solve because the objective is the ratio of largest eigenvalue λ_{max} over smallest nonzero eigenvalue λ_{min}. Instead, we choose to maximize smallest eigenvalue, while keeping largest eigenvalue bounded to avoid scale ambiguity. Alternatively, one can also minimize λ_{max} while keeping λ_{min} bounded. We use CVX to solve this problem. However, CVX does not accept quadratic terms as objective except for scalars, so $\lambda_{\text{min}} = \lambda_2(\boldsymbol{D} - \boldsymbol{H}\boldsymbol{H}^\top)$ cannot be optimized directly. To circumvent this obstacle, we relax the original problem and treat $\boldsymbol{W} = \boldsymbol{H}\boldsymbol{H}^\top$ as a new variable. Then problem (3.15) translates to

$$\begin{aligned} \max_{\boldsymbol{D}_w, \boldsymbol{W}} \quad & \lambda_2(\boldsymbol{D}_w - \boldsymbol{W}) \\ \text{s.t.} \quad & \boldsymbol{D}_w \in \mathcal{D}, \boldsymbol{W} \in \mathcal{W} \\ & \boldsymbol{W} \succeq 0 \\ & \boldsymbol{D}_w - \boldsymbol{W} \succeq 0 \\ & \text{Null}(\boldsymbol{D}_w - \boldsymbol{W}) = \text{span}\{\mathbf{1}\} \\ & \lambda_N(\boldsymbol{W}) \leq q \end{aligned} \tag{3.16}$$

where q is some constant, and $\mathcal{W} \in \mathbb{R}^{N \times N}$ denotes the set of matrices with the same sparsity pattern as $\boldsymbol{C}\boldsymbol{C}^\top$. If we choose q such that λ_{max} is equal or less than that $\lambda_N(\boldsymbol{C}\boldsymbol{E}\boldsymbol{C}^\top)$ of HADMM while maximizing λ_{min}, then WHADMM is guaranteed to have a larger rate round, which, in practice, often leads to faster convergence.

To find weights the underlying $\mathscr{G}$, we need to recover $\boldsymbol{H}$ from $\boldsymbol{W}$. Since $\boldsymbol{H} \in \mathcal{C}$, we obtain that $\text{rank}(W) = \text{rank}(H) = \min\{N, M\}$. By the way of constructing $\mathscr{H}$, we have $M \leq N$, thus $\boldsymbol{W} \in \mathbb{R}^{N \times N}$ has the same rank as $\boldsymbol{H}$. We can take advantage of this low-rank structure to recover $\boldsymbol{H}$. In particular, upon obtaining $\boldsymbol{W}$, we can recover $\boldsymbol{H}$ by solving a nonnegative

matrix factorization (NMF) problem

$$\begin{aligned} \min_{\boldsymbol{H}} \quad & \|\boldsymbol{H}\boldsymbol{H}^\top - \boldsymbol{W}\|_F^2 \\ \text{s.t.} \quad & \boldsymbol{H} \in \mathcal{C} \end{aligned} \tag{3.17}$$

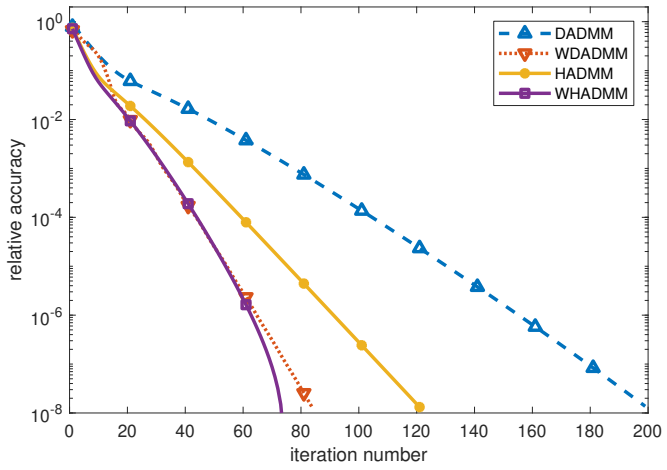
where $\boldsymbol{W}$ is the solution of (3.16). Problem (3.17) can solved by many NMF algorithms, but it turns out the simple projected gradient descent suffices.

3.6 Experiments

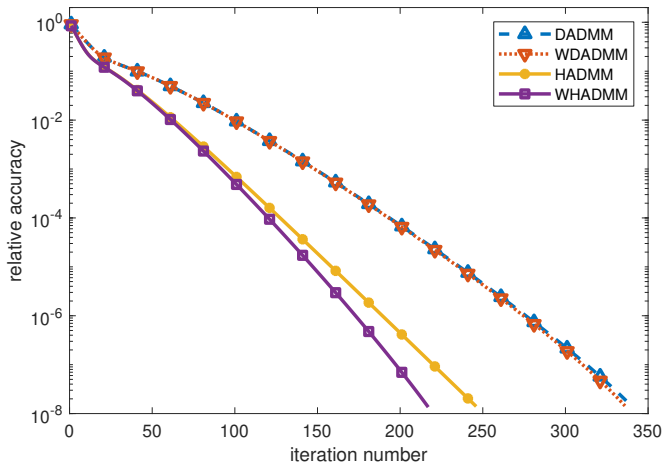
In this section, we compare the performance of WHADMM with existing algorithms, including HADMM, WDADMM and DADMM. The problem we are dealing with is network average, which can be formulated as a least squares problem $f(\boldsymbol{X}) = \|\boldsymbol{X} - \boldsymbol{O}\|_F^2/2$, where $\boldsymbol{O} = [\boldsymbol{o}_i^\top; \ldots; \boldsymbol{o}_N^\top] \in \mathbb{R}^{N\times p}$ collects nodal observations $\{\boldsymbol{o}_i\}$. The cost function is separable across nodes, i.e., $f(\boldsymbol{X}) = \sum_i f_i(\boldsymbol{x}_i)$ where $f_i(\boldsymbol{x}_i) = \|\boldsymbol{x}_i - \boldsymbol{o}_i\|_2^2$. Each nodal observation $\boldsymbol{o}_i$ is generated according to uniform distribution $U[0, N]$. We test the algorithms on two graphs, a graph with two clusters and a line graph. We set $N = 31$ for both graphs. Though there are adaptive ways to tune penalty parameter ρ, our focus in this experiment is to demonstrate convergence properties of four decentralized optimization algorithms, so we keep ρ fixed and tuned to optimal for each algorithm. We obtain edge weights by solving (3.16) and (3.17).

We measure the progress of algorithms by the number of iterations needed to attain certain solutions with relative error, $\|\boldsymbol{x} - \boldsymbol{x}^\star\|/\|\boldsymbol{x}^\star\|$. We plot relative accuracy against iteration number, shown in Fig. 3.2. In particular, Fig. 1(a) shows that for the graph with two clusters, both WHADMM and WDADMM have similar performance; HADMM, while slower than the former two, still significantly outperforms DADMM. Fig. 13.2b shows that WHADMM has similar convergence speed with HADMM, meaning tuning weights does not help much. Also, in this case, WDADMM and DADMM perform almost the same, confirming the ineffectiveness of weight tuning for line graphs. These observations conform with those of [65, 71]. In both cases, one can that WHADMM consistently better than WDADMM and HADMM, although this margin may be small for line graphs.

By inspecting the weights obtained by WHADMM, we make some interesting observations. Consider the graph in Fig. 3.1 with two clusters connected by a bridge node. The weights



(a) Convergence speed over a graph of two clusters.



(b) Convergence speed over a line graph.

Figure 3.2: Performance comparison of four decentralized algorithms over different graphs.

Table 3.1: Optimal weights obtained by solving (3.16) and (3.17).

left cluster		right cluster		middle	
edge	weight	edge	weight	edge	weight
(1, 1)	0.7905	(6, 6)	0.7905	(5, 5)	0.7923
(1, 2)	0.7905	(6, 7)	0.7905	(5, 11)	1.1814
(1, 3)	0.7905	(6, 8)	0.7905	(10, 11)	1.1814
(1, 4)	0.7905	(6, 9)	0.7905		
(1, 5)	0.9508	(6, 10)	0.9508		

obtained, presented in Table 3.1, show that WHADMM is able to identify critical edges, i.e., edges (1, 5), (6, 10), (5, 11) and (10, 11), and assign larger weights. Thus, sitting at node 1, updates from node 5 are regarded as more important than those from nodes 1, 2, 3 and 4, which partially explains why WHADMM can attain better performance.

3.7 Chapter summary

This chapter introduced WHADMM that is able to deal with weighted graphs and has the advantages of exploiting topology information to achieve better performance. By maximizing the convergence rate, the optimal weights obtained carry a meaningful interpretation, in the sense that critical edges are regarded as more important, aligned with intuition. Future works include exploring dropping edges by assigning zero weights in order to reduce communication overhead, and extending beyond ADMM to other optimization methods.

Chapter 4

Optimal Design of Weights of Decentralized Optimization via Preconditioning

4.1 Introduction

Consider the distributed optimization problem using N networked computing machines

$$\min_{\boldsymbol{x}} \sum_{i=1}^{N} f_i(\boldsymbol{x}) \tag{4.1}$$

where f_i is the local objective function only accessible to node i and $\boldsymbol{x} \in \mathbb{R}^l$ is the common decision variable. Distributed optimization problems arise when the computation or storage load is beyond the capability of a single machine, or data are collected at dispersed locations. Examples includes multi-agent coordination, large scale machine learning, distributed signal processing over sensor networks [93, 9, 37, 60].

Problem (4.1) is typically formulated in consensus form and then solved by consequently [82, 93, 83, 105, 9]. ADMM is particularly appealing for such problems thanks to the mild condition for convergence and its ability to decompose the objective into (easier) subproblems [5, 9]. Typically, ADMM-based algorithms alternate between a variables update step and a

communication step. Based on the communication pattern, these algorithms can be further classified as centralized ADMM (CADMM) [9, 5] or decentralized ADMM (DADMM) [100, 103]. The applicability of CADMM is quite limited due to its reliance on the star topology. Though DADMM can be used for any kind of graphs, it cannot adapt to the specific network topology.

The hybrid ADMM (HADMM) takes node importance into consideration by locally aggregating updates [71]. This chapter considers the weighted hybrid ADMM (WHADMM), which takes into account both node importance and edge weights. It turns out that accommodating edge weights is equivalent to preconditioning ADMM. Thus, optimal weight design amounts to finding the preconditioning matrix that yields the best performance.

There are some works that considers improving performance of ADMM via preconditioning [41, 118], but their results are shown for general ADMM, which only apply to decentralized optimization on star graphs. For other graphs, only a heuristic method for DADMM exists [65]. Our contribution is two-fold: i) we provide a preconditioning approach that works for any kind of graphs; and ii) our approach provides some optimality guarantee both for star graphs and general graphs.

Related works. A closely related work [54] considers cluster-based ADMM, but requires another subprocedure, e.g., gossip algorithm, to compute cluster related quantities. Decentralized ADMM over weighted graphs was also considered [65], which offers the flexibility to optimize convergence rate by tuning weights. The importance of edges was incorporated into HADMM as well [71] and heuristic method to compute edge weights are also provided, but without optimality guarantee. ADMM is equivalent to *Douglas-Rachford splitting* (DRS) applied to the dual problem [22]. Therefore, one can prove convergence of ADMM through Douglas-Rachford algorithm. Recently, a tight linear rate of ADMM has been established, and preconditioning approaches to improve the convergence speed is discussed [41, 40]. These results, however, entails assumptions that are difficult to satisfy in practice.

4.2 Decentralized optimization

The decentralized optimization problem aims to solve (4.1) using N networked computing machines whose connectivity is specified by some *communication graph* $\mathscr{G}$. To decouple the problem, each machine has its own local decision variable, and neighbors must agree on common

values, which translates to formulation

$$\begin{aligned} \min_{\{\boldsymbol{x}_i\}} \quad & \sum_{i=1}^{N} f_i(\boldsymbol{x}_i) \\ \text{subject to} \quad & \boldsymbol{x}_i = \boldsymbol{x}_j, \quad (i,j) \text{ specified by } \mathscr{G} \end{aligned} \tag{4.2}$$

where $\boldsymbol{x} = (\boldsymbol{x}_1^\mathsf{T}, \boldsymbol{x}_2^\mathsf{T}, \ldots, \boldsymbol{x}_N^\mathsf{T})^\mathsf{T}$ collects all local variables into one vector. The decoupled problem allows parallel minimization of each local cost function, and a valid solution can be obtained by periodically exchanging information among computing nodes to ensure the whole network reaches consensus.

Communication graph. Communication among computing machines, however, is not always possible, due to limitations such as location, physical connectivity, or simply privacy concerns. These constraints can be conveniently specified by a *communication graph* $\mathscr{G} := (\mathcal{V}, \mathcal{E})$, where the node set $\mathcal{V} := \{1, 2, \ldots, N\}$ describes the labels of all computing nodes, and the edge set $\mathcal{E} := \{\mathcal{E}_1, \ldots, \mathcal{E}_M\}$ includes tuples of all connected nodes. Notice that a tuple does not necessarily includes two nodes. For example, the communication graph of DADMM is an undirected graphs, where each node corresponds to one computing machine and each edge $\mathcal{E}_j$ is a tuple of *exactly* two nodes that can exchange information. Such a communication graph leads to linear constraints $\boldsymbol{x}_n = \boldsymbol{x}_m$ for all $(i,j) \in \mathcal{E}$, which becomes $\boldsymbol{x}_i = \boldsymbol{z}_{ij}, \boldsymbol{x}_j = \boldsymbol{z}_{ij}$ upon introducing an auxiliary variable $\boldsymbol{z}_{ij}$ per edge.

The communication graph does not necessarily be a simple graph. For example, the case when more than two nodes can exchange information at once cannot be described by a simple edge. Instead, we introduce hypergraphs to cope with such situations. A hypergraph differs from a simple graph that each hyperedge can include multiple nodes, i.e., it is possible to have $|\mathcal{E}_j| \geq 2$.

The *hyperedge* is a generalization of *neighborhood* relation, and nodes residing in the same group can exchange information with any other, thus becoming *neighbors*. We have $\bigcup_{j=1}^{M} \mathcal{E}_j = \mathcal{V}$, where M is the total number of hyperedges. Using hypergraphs as the communication graph was introduced in [71]. The use of hypergraphs provides a unifying view of ADMM-based distributed optimization. For example, there is only one hyperedge containing all nodes in CADMM; while for DADMM, each hyperedge edge reduces to one simple edge consisting of the two connected nodes. i.e., $|\mathcal{E}_j| = 2, \forall j$.

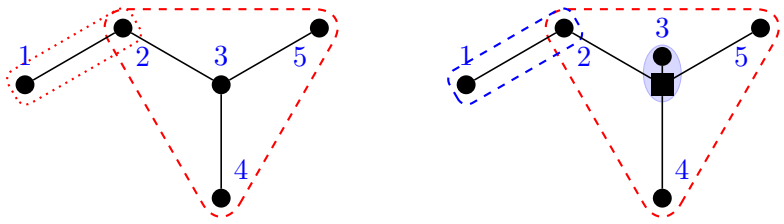
Important matrices. Assume an arbitrary order of hyperedges and denote j-th group as $\mathcal{E}_j \subset \mathcal{V}$. Assign an auxiliary variable per hyperedge and impose equality constraints inside each hyperedge. The linear constraints in formulation (4.2) becomes $\boldsymbol{x}_i = \boldsymbol{z}_j,\ i \in \mathcal{E}_j$ where $\boldsymbol{z}_j$ is the auxiliary variable assigned to the j-th hyperedge. Such constraints are mathematically equivalent to those of (4.2), provided that the communication graph is connected.

Writing the constraints in matrix-vector form reveals the intrinsic structure of this problem, upon understanding some properties of matrices. Specifically, we can write all hypergraph-based constraints $\boldsymbol{x}_i = \boldsymbol{z}_j,\ i \in \mathcal{E}_j$ into one linear equation $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{B}\boldsymbol{z}$, where $\boldsymbol{x} \in \mathbb{R}^{Nl}$ is the vector obtained by concatenating all $\boldsymbol{x}_i$ and $\boldsymbol{z} \in \mathbb{R}^{Ml}$ by concatenating all $\boldsymbol{z}_j$. The rows of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ essentially select the corresponding $\boldsymbol{x}_i$ and $\boldsymbol{z}_j$ for each constraint. Let T denote the total number of constraints and label all constraints in arbitrary order. Then $\boldsymbol{A} \in \mathbb{R}^{Tl\times Nl}$ and $\boldsymbol{B} \in \mathbb{R}^{Tl\times Ml}$ are defined such that if the t-th constraint is $\boldsymbol{x}_i = \boldsymbol{z}_j$, then (t,i)-th block of $\boldsymbol{A}$ and (t,j)-th block of $\boldsymbol{B}$ are identity matrices; all other blocks in the t-th row are zeros.

One important matrix of an hyperedge is the (signless) incidence matrix $\boldsymbol{C} \in \mathbb{R}^{Nl\times Ml}$, whose (i,j)-th block is $\boldsymbol{I}$ if i-th node belongs to j-hyperedge, and zeros otherwise. Similar to the degree of a node, the degree of an hyperedge $\mathcal{E}_j$ can be defined as the number of nodes in $\mathcal{E}_j$, which is also the cardinality $|\mathcal{E}_j|$. Define block diagonal matrices $\boldsymbol{D} := \operatorname{diag}(d_1,\dots,d_N)\otimes\boldsymbol{I}$, $\boldsymbol{E} := \operatorname{diag}(e_1,\dots,e_M)\otimes\boldsymbol{I}$, where d_i is the degree of i-th node and e_j the degree of j-th hyperedge. Then the following relations hold: $\boldsymbol{A}^\mathsf{T}\boldsymbol{A} = \boldsymbol{D}, \boldsymbol{B}^\mathsf{T}\boldsymbol{B} = \boldsymbol{E}, \boldsymbol{A}^\mathsf{T}\boldsymbol{B} = \boldsymbol{C}$ ([71, Lemma 1]).

Example 2. Consider a decentralized optimization problem over a graph shown in Fig. 4.1(a) with $l = 1$. There are 4 nodes and 2 hyperedges, namely $\mathcal{E}_1 := \{1,2\}$ and $\mathcal{E}_2 := \{2,3,4,5\}$, shown as gray dashed circles. Clearly, this grouping scheme results in a connected hypergraph. Based on the current grouping, the matrices related to this hypergraph are given by

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{B} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \boldsymbol{D} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{E} = \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix} \boldsymbol{C} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}.$$



(a) (b)

Figure 4.1: A simple graph illustrating one possible grouping. Solid black circles represent nodes with their numbering, solid lines indicate connectivity between nodes, and dashed circles identify groups.

One can verify that the relations among important matrices indeed hold.

Weighted graphs. Not all edges are equally important. It is often necessary to take into consideration the relative importance of edges while designing algorithms for decentralized optimization, either because the network edges are weighted, or performance improvement can be obtained by carefully designing weights. Either way, one needs to incorporate weight information into problem formulation. It turns out that weight information can be represented by a diagonal scaling matrix left multiplied to linear constraint $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{B}\boldsymbol{z}$. Let $\boldsymbol{S} \in \mathbb{R}^{Tl \times Tl}$ be the scaling matrix such that $\boldsymbol{S}^2 = \mathrm{diag}(w_1, w_2, \ldots, w_T) \otimes \boldsymbol{I}$, where w_t is the weight associated with t-th constraint. Then all the important matrices aforementioned have corresponding definitions, and the relations among them still hold. For example, $\hat{\boldsymbol{A}} = \boldsymbol{S}\boldsymbol{A}, \hat{\boldsymbol{B}} = \boldsymbol{S}\boldsymbol{B}$, and $\hat{\boldsymbol{D}} = \hat{\boldsymbol{A}}^\top \hat{\boldsymbol{A}} = \boldsymbol{A}^\top \boldsymbol{S}^\top \boldsymbol{S}\boldsymbol{A}$.

Problem formulation. Accounting for edge weight information, problem (4.1) can be formulated as

$$\begin{aligned} \min_{\{\boldsymbol{x}_i, \boldsymbol{z}_j\}} \quad & \sum_{i=1}^{n} f_i(\boldsymbol{x}_i) \\ \text{s. to} \quad & \hat{\boldsymbol{A}}\boldsymbol{x} = \hat{\boldsymbol{B}}\boldsymbol{z} \end{aligned} \tag{4.3}$$

where $\hat{\boldsymbol{A}}, \hat{\boldsymbol{B}}$ are defined according to the underlying communication graph.

4.3 Weighted hybrid ADMM

Problem (4.3) is ready to be solved using ADMM. Let $\lambda_t \in \mathbb{R}^l$ be the Lagrange multiplier of the t-th constraint, and $\lambda = (\lambda_1^\mathsf{T}, \ldots, \lambda_T^\mathsf{T})^\mathsf{T} \in \mathbb{R}^{Tl}$ collects all multipliers. ADMM iterations for solving (4.3) can be carried out with $\boldsymbol{x}, \lambda$ only, with $\boldsymbol{z}$ completed eliminated.

Proposition 4. *Suppose f is convex and the Lagrange of* (4.3) *admits at least one saddle point. Upon initializing λ^0 such that $\boldsymbol{B}^\mathsf{T}\lambda^0 = \mathbf{0}$, then ADMM iterations for solving* (4.3) *can be simplified as*

$$\begin{aligned} \boldsymbol{x}^{k+1} &= \arg\min_{\boldsymbol{x}} f(\boldsymbol{x}) + \frac{\rho}{2}\boldsymbol{x}^\mathsf{T}\hat{\boldsymbol{D}}\boldsymbol{x} + \boldsymbol{x}^\mathsf{T}(\boldsymbol{y}^k - \gamma\hat{\boldsymbol{C}}\hat{\boldsymbol{E}}^{-1}\hat{\boldsymbol{C}}^\mathsf{T}\boldsymbol{x}^k) \\ \boldsymbol{y}^{k+1} &= \boldsymbol{y}^k + \gamma(\hat{\boldsymbol{D}} - \hat{\boldsymbol{C}}\hat{\boldsymbol{E}}^{-1}\hat{\boldsymbol{C}}^\mathsf{T})\boldsymbol{x}^{k+1} \end{aligned} \tag{4.4}$$

where $\boldsymbol{y}^k = \boldsymbol{A}^T\lambda^k$ is a change of variable, γ a penalty parameter and k iteration index.

The ADMM iterations are fully separable across nodes and edges, thanks to the structures of $\hat{\boldsymbol{D}}, \hat{\boldsymbol{E}}$ and $\hat{\boldsymbol{C}}$. Towards this end, notice that $\hat{\boldsymbol{D}}$ is block diagonal, thus both $\hat{\boldsymbol{D}}\boldsymbol{x}$ and $\boldsymbol{x}^\mathsf{T}\hat{\boldsymbol{D}}\boldsymbol{x}$ are fully separable across nodes. It remains to show $\hat{\boldsymbol{C}}\hat{\boldsymbol{E}}^{-1}\hat{\boldsymbol{C}}^\mathsf{T}\boldsymbol{x}$ is also separable. To simply notation, let $l = 1$. Let $\boldsymbol{u} = \hat{\boldsymbol{C}}^\mathsf{T}\boldsymbol{x}$, then the i-th block of $\boldsymbol{u}$ is $\boldsymbol{u}_j = \sum_{i=1}^n \hat{c}_{ij}\boldsymbol{x}_i = \sum_{i\in\mathcal{E}_j} \hat{c}_{ij}\boldsymbol{x}_i$. Consequently, let $\boldsymbol{v} = \hat{\boldsymbol{C}}\boldsymbol{z} = \hat{\boldsymbol{C}}\hat{\boldsymbol{E}}^{-1}\hat{\boldsymbol{C}}^\mathsf{T}\boldsymbol{x}$, i-th row of $\boldsymbol{v}$ can be obtained by $\boldsymbol{v}_i = \sum_{j=1}^M \hat{c}_{ij}\boldsymbol{z}_j = \sum_{i\in\mathcal{E}_j} \hat{c}_{ij}\boldsymbol{z}_j$ which is a weighted sum of $\boldsymbol{z}_j$ from all connected coordinators. Therefore, $\boldsymbol{v}_i$ can be computed by node i through communication with neighboring nodes and coordinators. One can show that ADMM iterations at node i are

$$\begin{aligned} \boldsymbol{x}_i^{k+1} &= \arg\min_{\boldsymbol{x}_i} f_i(\boldsymbol{x}_i) + \frac{\rho}{2}d_i\|\boldsymbol{x}_i\|^2 + \boldsymbol{x}_i^\mathsf{T}(\boldsymbol{y}_i^k - \rho\boldsymbol{v}_i^k) \\ \boldsymbol{y}_i^{k+1} &= \boldsymbol{y}_i^k + \rho(d_i\boldsymbol{x}_i^{k+1} - \boldsymbol{v}_i^{k+1}). \end{aligned} \tag{4.5}$$

From (4.5) we can see that node i only needs $\boldsymbol{v}_i$, which can be computed by communicating with neighbors, to finish $(k+1)$-th iteration.

The matrix $\hat{\boldsymbol{L}_\mathscr{G}} = \hat{\boldsymbol{D}} - \hat{\boldsymbol{C}}\hat{\boldsymbol{E}}^{-1}\hat{\boldsymbol{C}}^\mathsf{T}$ is a valid graph Laplacian, i.e., $\hat{\boldsymbol{L}_\mathscr{G}}\mathbf{1} = \mathbf{0}$ [71]. In fact, it is the Laplacian of the underlying communication graph, which has been investigated in [7].

Fully decentralized implementation. The ADMM iterations (4.4) are fully separable across nodes and edges, thanks to the structures of $\hat{\boldsymbol{D}}, \hat{\boldsymbol{E}}$ and $\hat{\boldsymbol{C}}$. However, to implement this algorithm in fully decentralized manner, one important question remains: *which node is*

responsible for computing $\boldsymbol{z}_j$? Intuitively, a dedicate machine, different from a normal computing machine that has a local cost f_i, is needed to update $\boldsymbol{z}_j$. But this cannot be implemented using current available machines. Using the technique termed *in-network acceleration* [71], we can implement the iterations using only N networked nodes. For example, in Fig. 4.1(b), the right hyperedge consists of four nodes and the center is node 3. We can create a virtual coordinator inside node 3, shown by a black square, and connect it to node 2, 4 and 5 using existing edges. Node 3 plays two roles: a normal node 3 and a coordinator. Thus, WHADMM iterations can be carried out without dedicated machines. This also suggest a heuristic method of creating hypergraphs. One should select nodes with high degrees and create hyperedges containing those nodes and all their neighbors. This method of creating hypergraphs implicitly uses node degree centrality as a metric to place coordinators.

4.3.1 Linear convergence rate

Proposition 5. *If f is α-strongly convex and has β-Lipschitz continuous gradient, and $\gamma = (\alpha\beta\lambda_{max}(\boldsymbol{L}_{\mathscr{G}}))/(\|\boldsymbol{A}\|2(\lambda_{min>0}(\boldsymbol{L}_{\mathscr{G}}))^2)$, then the sequence $\{\lambda\}$ converges geometrically towards optimal value with rate δ, that is,*

$$\|\lambda^k - \lambda^\star\| \leq \delta^k C \tag{4.6}$$

where C is some constant, $\delta = \sqrt{\kappa_d}/(\sqrt{\kappa_d}+1)$, and

$$\kappa_d = \frac{\beta}{\alpha}\frac{\|\boldsymbol{A}\|^2\lambda_{max}(\boldsymbol{L}_{\mathscr{G}})}{(\lambda_{min>0}(\boldsymbol{L}_{\mathscr{G}}))^2}. \tag{4.7}$$

Proposition 5 establishes the linear convergence of ADMM through DRS applied to the dual problem, and provides explicit formula for this rate. We can see that this rate depends on the condition number of objective β/α, the maximum node degree $\|\boldsymbol{A}\|^2$, and spectral property of the graph Laplacian.

Tightness of the bound. It has been shown that the linear convergence rate in proposition 5 is tight when $\boldsymbol{A}$ has full row rank, i.e., there exist a certain class of problems satisfying the assumption can actually achieve this rate of convergence, see [41] for proof and examples. This immediately implies the rate in proposition 5 is also tight. In other words, there exist certain problems that this rate can actually be achieved.

4.4 Optimal design of weights

For decentralized optimization, weighted hybrid ADMM amounts to preconditioning the linear constraint using a diagonal scaling matrix. Thus the optimal design of weights amounts to finding the best scaling matrix that maximizes the convergence rate of ADMM. Without loss of generality, we consider the case $l = 1$ in this section.

4.4.1 Preconditioned ADMM

For a positive definite matrix $\boldsymbol{P}$, define the weighted norm induced by $\boldsymbol{P}$ as $\|\boldsymbol{x}\|_{\boldsymbol{P}}^2 = \boldsymbol{x}^\mathsf{T}\boldsymbol{P}\boldsymbol{x}$. A function f is α-strongly convex if and only if $f - \|\cdot\|^2/2$ is convex; f is β-smooth if and only if $\|\cdot\|^2/2 - f$ is convex. Define the preconditioned function $f_S(\boldsymbol{x}) := f(\boldsymbol{S}\boldsymbol{y})$ and let $\boldsymbol{H}, \boldsymbol{M}$ be two positive definite matrices. Lemma 6 characterizes the properties of the preconditioned functions.

Lemma 6 ([39, Proposition 7])**.** *If a closed, convex, and proper function f is $\lambda_{min}(\boldsymbol{H})$-strongly convex and $\lambda_{max}(\boldsymbol{M})$-smooth, then $f_S(\boldsymbol{x})$ is $\lambda_{min}(\boldsymbol{S}\boldsymbol{H}\boldsymbol{S}^\mathsf{T})$-strongly convex and $\lambda_{max}(\boldsymbol{S}\boldsymbol{H}\boldsymbol{S}^\mathsf{T})$-smooth.*

Since the tight linear convergence rate of ADMM is obtained through the equivalence of ADMM and DRS applied to the dual [22], we need to characterize the properties of the dual function. Consider the dual problem of (4.3)

$$\begin{aligned} &\min_{\lambda} \; d(\lambda) = f^*(-\hat{\boldsymbol{A}}^\mathsf{T}\lambda) \\ &\text{s. to} \;\; \hat{\boldsymbol{B}}^\mathsf{T}\lambda = \boldsymbol{0} \end{aligned} \tag{4.8}$$

where f^* is the conjugate function of f.

Lemma 7 ([41, Proposition 5])**.** *If f is $\lambda_{min}(\boldsymbol{H})$-strongly convex and $\lambda_{max}(\boldsymbol{M})$-smooth, and $\boldsymbol{A}$ has full row rank, then the dual d is $\lambda_{min}(\boldsymbol{A}\boldsymbol{M}^{-1}\boldsymbol{A}^\mathsf{T})$-strongly convex and $\lambda_{max}(\boldsymbol{A}\boldsymbol{H}^{-1}\boldsymbol{A}^\mathsf{T})$-smooth.*

4.4.2 Optimal scaling matrix

The convergence analysis of weighted hybrid ADMM reveals the dependence of convergence rate on spectral properties of the underlying graph. When modifying graph topology is possible,

one can always select a *central node* and connect it to all others to create a star graph, which yields the best performance. When it is impossible to adjust the connections among nodes, we can compute optimal weights that leads to optimal convergence rate. Typically, we do not have the freedom to change network topology, thus the option left is to optimally design the weights. Moreover, when the edge weights are already given but not optimal, we can always compute the best weights and properly scale the given weights. We distinguish two cases in the consequent discussion based on whether or not $\boldsymbol{A}$ has full row rank.

Case I: $\boldsymbol{A}$ has full row rank

When $\boldsymbol{A}$ is full row rank, proposition 5 and the results of [41] are equivalent, and we choose the latter for its simple form. It can be shown that the linear convergence rate of preconditioned ADMM when solving decentralized optimization (4.3) depends on the ratio

$$\lambda_{\max}(\boldsymbol{S}\boldsymbol{A}\boldsymbol{H}^{-1}\boldsymbol{A}^{\mathsf{T}}\boldsymbol{S}^{\mathsf{T}})/\lambda_{\min}(\boldsymbol{S}\boldsymbol{A}\boldsymbol{M}^{-1}\boldsymbol{A}^{\mathsf{T}}\boldsymbol{S}^{\mathsf{T}}). \tag{4.9}$$

Smaller ratio leads to faster rate [41]. The optimal design of weights is thus obtained by solving the optimization problem

$$\min_{\boldsymbol{S}\in\mathcal{S}} \quad \frac{\lambda_{\max}(\boldsymbol{S}\boldsymbol{A}\boldsymbol{H}^{-1}\boldsymbol{A}^{\mathsf{T}}\boldsymbol{S}^{\mathsf{T}})}{\lambda_{\min}(\boldsymbol{S}\boldsymbol{A}\boldsymbol{M}^{-1}\boldsymbol{A}^{\mathsf{T}}\boldsymbol{S}^{\mathsf{T}})} \tag{4.10}$$

where $\mathcal{S}$ denotes the set of all $T \times T$ positive definite diagonal matrices. This ratio arises in a lot of control problems, see e.g. [113]. Problem (4.10) can be expressed as an generalized eigenvalue problem that can be solved efficiently, see e.g. [10, 11].

Case II: $\boldsymbol{A}$ is row rank deficient

When $\boldsymbol{A}$ does not have full row rank, the convergence result proposition 5 does not hold anymore. This happens when a single group cannot cover all nodes. In this case, the diagonal scaling technique fails as $\boldsymbol{A}\boldsymbol{M}^{-1}\boldsymbol{A}^{\mathsf{T}}$ is rank deficient and $\lambda_{\min}(\boldsymbol{A}\boldsymbol{M}^{-1}\boldsymbol{A}^{\mathsf{T}}) = 0$, leaving the ratio undefined. So the direct minimization of the cross condition number will not work. We introduce two preconditioning approaches to circumvent this issue.

Approach I. One could resort to a heuristic approach by considering a quantity closed related to the cross condition number, $\lambda_{\max}(\boldsymbol{A}\boldsymbol{H}^{-1}\boldsymbol{A}^{\mathsf{T}})/\lambda_{\min>0}(\boldsymbol{A}\boldsymbol{M}^{-1}\boldsymbol{A}^{\mathsf{T}})$ instead, hoping that the scaling that minimizes this quantity would yield well conditioned problem and better

performance. That being said, we can find the scaling by solving

$$\min_{\boldsymbol{S}\in\mathcal{S}} \quad \frac{\lambda_{\max}(\boldsymbol{S}\boldsymbol{A}\boldsymbol{H}^{-1}\boldsymbol{A}^{\mathsf{T}}\boldsymbol{S}^{\mathsf{T}})}{\lambda_{\min>0}(\boldsymbol{S}\boldsymbol{A}\boldsymbol{M}^{-1}\boldsymbol{A}^{\mathsf{T}}\boldsymbol{S}^{\mathsf{T}})}. \tag{4.11}$$

Again, algorithms for solving this problem can be found in [10, 40]. Though it works in practice, as demonstrated in numerical experiments in the sequel, this method lacks some theoretical ground to stand upon. The best one can do is to try it out and hope that it will improve the performance.

Approach II. For decentralized optimization problems, the cases $\boldsymbol{A}$ has full row rank arise only when there exists a *central node* connected to all others, which can serve as a center to create a virtual coordinator. Depending on the requirement, we may not be able to modify existing graph topology by adding or removing edges. Thus, we may find ourselves frequently dealing with the cases where $\boldsymbol{A}$ does not have full row rank and [41] fails to provide a valid theoretical bound. As a result, we have to resort to Approach I as a heuristic method.

Since proposition 5 holds for the cases where $\boldsymbol{A}$ does not have full row rank, one can alternatively try to find a diagonal scaling matrix that maximize the rate bound in proposition 5. This bound depends on spectral radius of $\boldsymbol{A}$, which is equivalent to the maximum node degree, and the ratio of largest and smallest nonzero eigenvalue of Laplacian $\boldsymbol{L}_{\mathscr{G}}$. Trying to optimize this rate directly is difficult as it is a nonconvex nonsmooth problem [10].

Instead of minimizing the rate as a whole, we choose to optimize the condition number of the communication graph, defined as the ratio between the largest eigenvalue of Laplacian $\lambda_{\max}(\boldsymbol{L}_{\mathscr{G}})$ and the smallest nonzero eigenvalue $\lambda_{\min>0}(\boldsymbol{L}_{\mathscr{G}})$, which also optimizes the bound. Towards this end, notice that the degree of a central node reduces to one once an hyperedge is created using its neighborhood. A grouping strategy that prioritizes higher degree nodes would automatically takes care of $\|\boldsymbol{A}\|_2$. At the same time, minimizing the condition number of communication graph also decreases $\lambda_{\max}(\boldsymbol{L}_{\mathscr{G}})/\lambda_{\min>0}(\boldsymbol{L}_{\mathscr{G}})^2$.

Minimizing the condition number of the communication graphs is equivalent to solving the optimization problem

$$\min_{\boldsymbol{S}\in\mathcal{S}} \quad \frac{\lambda_{\max}(\hat{\boldsymbol{L}_{\mathscr{G}}})}{\lambda_{\min>0}(\hat{\boldsymbol{L}_{\mathscr{G}}})} \tag{4.12}$$

where $\hat{\boldsymbol{L}_{\mathscr{G}}} = \hat{\boldsymbol{D}} - \hat{\boldsymbol{C}}\hat{\boldsymbol{E}}^{-1}\hat{\boldsymbol{C}}^{\mathsf{T}} = \hat{\boldsymbol{A}}^{\mathsf{T}}\boldsymbol{S}^{\mathsf{T}}\boldsymbol{S}\hat{\boldsymbol{A}} - \hat{\boldsymbol{A}}^{\mathsf{T}}\boldsymbol{S}^{\mathsf{T}}\boldsymbol{S}\hat{\boldsymbol{B}}(\hat{\boldsymbol{B}}\boldsymbol{S}^{\mathsf{T}}\boldsymbol{S}\hat{\boldsymbol{B}})^{-1}\hat{\boldsymbol{B}}\boldsymbol{S}^{\mathsf{T}}\boldsymbol{S}\hat{\boldsymbol{A}}$. Unfortunately, this problem is nonconvex and nonsmooth in $\boldsymbol{S}$, and cannot be solved easily. Leveraging

the special structure of $\boldsymbol{L}_{\mathscr{G}}$, we can convert problem (4.12) to one that is much easier to deal with.

Lemma 8. *Let $\boldsymbol{G} = \boldsymbol{A}\boldsymbol{A}^{\mathsf{T}} + \boldsymbol{B}\boldsymbol{B}^{\mathsf{T}}$. Then $\boldsymbol{G}$ is positive semidefinite, and we have*

$$\operatorname{rank}(\boldsymbol{G}) = \operatorname{rank}(\boldsymbol{E}) + \operatorname{rank}(\boldsymbol{L}_{\mathscr{G}}) = M + N - 1. \tag{4.13}$$

Lemma 8 shows that $\boldsymbol{G}$ is positive semidefinite with $M + N - 1$ nonzero eigenvalues, thus we arrive at $\lambda_{M+N-1}(\boldsymbol{G}) = \lambda_{\min>0}(\boldsymbol{G}) > 0$.

Theorem 4. *Let $\boldsymbol{G} = \boldsymbol{A}\boldsymbol{A}^{\mathsf{T}} + \boldsymbol{B}\boldsymbol{B}^{\mathsf{T}}$. Then the eigenvalues of $\boldsymbol{L}_{\mathscr{G}}$ interlace with those of $\boldsymbol{G}$, that is,*

$$\lambda_i(\boldsymbol{G}) \geq \lambda_i(\boldsymbol{L}_{\mathscr{G}}) \geq \lambda_{i+M}(\boldsymbol{G}), \quad i = 1, 2, \ldots, N-1. \tag{4.14}$$

Theorem 4 shows that the eigenvalues of $\boldsymbol{L}_{\mathscr{G}} \in \mathbb{R}^{n\times n}$ are bounded by a slightly larger matrix $\boldsymbol{G} \in \mathbb{R}^{t\times t}$. As discussed later, the benefit of working with a larger matrix $\boldsymbol{G}$ is that we can transform the highly nonconvex nonsmooth problem (4.12) into a standard condition number minimization problem. To show that, we first relates the condition number of $\boldsymbol{L}_{\mathscr{G}}$ to that of $\boldsymbol{G}$.

Corollary 2. The largest and smallest nonzero eigenvalues of $\boldsymbol{L}_{\mathscr{G}}$ are bounded by those of $\boldsymbol{G}$, i.e.,

$$\lambda_{\max}(\boldsymbol{G}) \geq \lambda_{\max}(\boldsymbol{L}_{\mathscr{G}}) \geq \ldots \geq \lambda_{\min>0}(\boldsymbol{L}_{\mathscr{G}}) \geq \lambda_{\min>0}(\boldsymbol{G}). \tag{4.15}$$

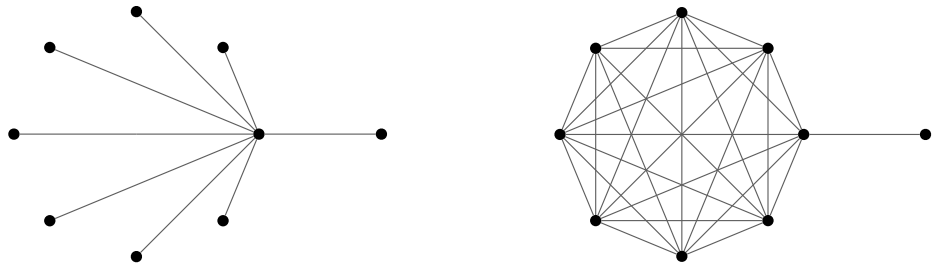
A direct consequence of corollary 2 is the condition number of $\boldsymbol{L}_{\mathscr{G}}$ is upper bounded by that of $\boldsymbol{G}$

$$\kappa(\boldsymbol{G}) \geq \kappa(\boldsymbol{L}_{\mathscr{G}}). \tag{4.16}$$

As a result, we can minimize the $\kappa(\boldsymbol{G})$ by choosing the proper scaling matrix $\boldsymbol{S}$. What makes this approach particularly appealing is that the size of $\boldsymbol{G}$ exactly matches the size of $\boldsymbol{S}$. In addition, minimizing the condition number of $\boldsymbol{G}$ by left and right multiplying scaling matrix $\boldsymbol{S}$ coincide with preconditioning. This becomes apparent upon realizing

$$\boldsymbol{S}\boldsymbol{G}\boldsymbol{S}^{\mathsf{T}} = \boldsymbol{S}\boldsymbol{A}(\boldsymbol{S}\boldsymbol{A})^{\mathsf{T}} + \boldsymbol{S}\boldsymbol{B}(\boldsymbol{S}\boldsymbol{B})^{\mathsf{T}}. \tag{4.17}$$

Therefore, this approach offers a meaningful interpretation of $\boldsymbol{S}$ since every diagonal element of $\boldsymbol{S}$ can be directly mapped to weight of each linear constraint $\boldsymbol{x}_i = \boldsymbol{z}_j$. With corollary 2, we



(a) (b)

Figure 4.2: The graph considered in Case I. In both cases, the communication graph is a hypergraph with only one hyperedge consisting of all nodes, based on which $\boldsymbol{A}$ has full row rank. Fig. (b) is obtained by adding some edges to (a).

have transformed the problem (4.12) to

$$\min_{\boldsymbol{S}\in\mathcal{S}} \quad \frac{\lambda_{\max}(\boldsymbol{S}\boldsymbol{G}\boldsymbol{S}^{\mathsf{T}})}{\lambda_{\min>0}(\boldsymbol{S}\boldsymbol{G}\boldsymbol{S}^{\mathsf{T}})}.. \tag{4.18}$$

Notice that $\boldsymbol{G}$ is positive semidefinite and might be singular. Problem (4.18) is actually a generalized eigenvalue problem [10], and can be solved using off-the-shelves methods [40, 10, 11, 113].

4.5 Numerical experiments

We perform numerical experiments to test the effects of different preconditioning approaches in case of $\boldsymbol{A}$ with full row rank and cases without. In particular, we consider the weighted network average problem with local observation $\boldsymbol{b}_i \in \mathbb{R}^3$. Let $\boldsymbol{x}_i$ be the decision variable of node i. The local objective function is $\frac{q_i}{2}\|\boldsymbol{x}_i - \boldsymbol{b}_i\|^2$, where q_i is the weight, and the overall objective is $f(\boldsymbol{x}) = \sum_{i=1}^N f_i(\boldsymbol{x}_i) = \frac{1}{2}(\boldsymbol{x}-\boldsymbol{b})^{\mathsf{T}}\boldsymbol{Q}(\boldsymbol{x}-\boldsymbol{b})$ where $\boldsymbol{Q} = \operatorname{diag}(q_1,\ldots,q_N)\otimes \boldsymbol{I}_3$. where $\boldsymbol{x}$ collects all $\boldsymbol{x}_i$ and $\boldsymbol{b} = (b_1, b_2, \ldots, b_n)^{\mathsf{T}}$. We measure the progress by the relative distance to optimal solution, $\|\boldsymbol{x}-\boldsymbol{x}^\star\|/\|\boldsymbol{x}^\star\|$, provided $\|\boldsymbol{x}^\star\| \neq 0$. Though proposition 5 provides the "best" γ in theory, we observe that it is not always the best in practice, especially when $\boldsymbol{A}$ is row rank deficient and the rate bound is not tight. So instead, we choose the best γ using grid search.

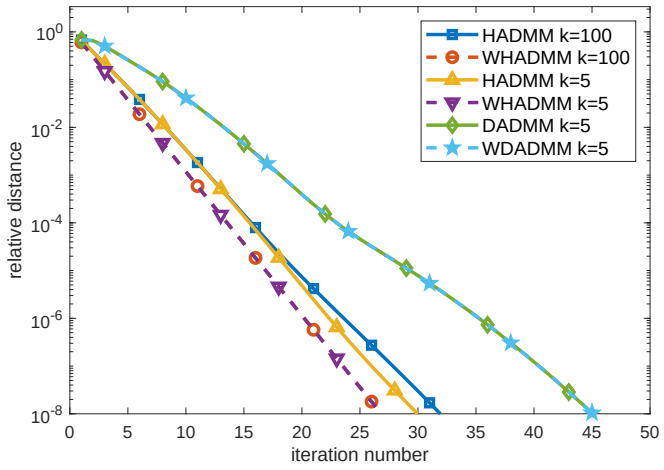
Case I: $\boldsymbol{A}$ full row rank. We first consider the case when $\boldsymbol{A}$ has full row rank. This happens

when all the nodes be connected in a single group, for example a star graph or a lollipop graph with all but one in a clique. We test over both graphs with the same number of nodes and the results are shown in Fig. 4.3 (a) and (b), respectively. We not only compare HADMM/WHADMM with DADMM/WDADMM, but also include the algorithm performance for different condition number of objective functions. Notice that in both cases, HADMM/WHADMM consistently outperforms DADMM/WDADMM, which demonstrates the benefits of hybrid method.

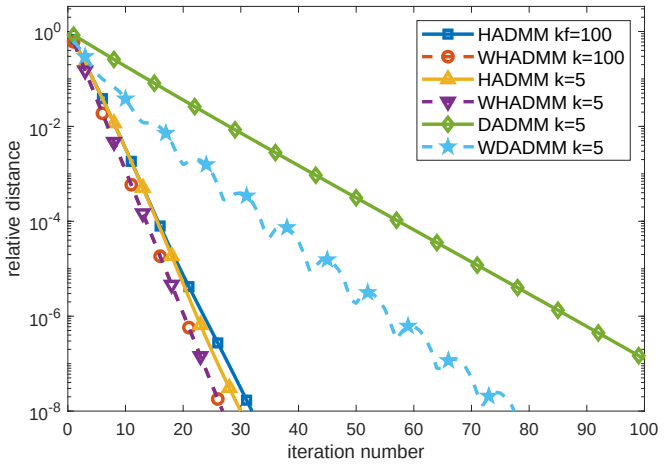
The gap is much larger in lollipop case. Interestingly, the lollipop has much more edges but yields to worse performance for DADMM and WDADMM. Intuitively, more edges should imply faster flow of information, hence faster convergence rate. However, the results imply that more edges not always leads to faster convergence. Since the preconditioning matrix can be solved exactly in this case, we expect to see the same performance regardless of condition number and graphs, which is corroborated by comparing two figures. Notice performance of both DADMM and WDADMM degrade greatly when the underlying graph changes from star to lollipop.

Case II: A row rank deficient. We consider a graph of two clusters connected by a single path, without which it becomes disconnected. The path is critical to this graph, and adjusting its weight can influence overall connectivity. To check the sensitivity to topology changes, we perform tests over two graphs, shown in Fig. 4.4. The first figure shows a graph of two clusters, each with 20 nodes, with 1 overlapping. The second graph is almost the same, with two nodes on the path, making the path 2 times longer from center to center. Both graphs have the same number of nodes. The test results can be found in Fig. 4.4(c) and Fig. 4.4(d).

A glance of both figures reveals that hybrid methods outperform fully decentralized ones. Moreover, one can see that WDADMM performs much better than DADMM, even better than WHADMM with Approach I, thus validating that properly designed weights can be crucial for DADMM. Comparing Fig. 4.4 (c) and (d), we see that without preconditioning, a small change in graph topology can lead to huge performance difference in DADMM and HADMM. Preconditioned HADMM achieves almost the same convergence speed for both graphs. This clearly demonstrates that preconditioning can greatly improve the robustness to topology changes. Also notice that preconditioning approach I consistently yields a worse convergence speed, even worse than HADMM in Fig. 4.4, showing that it is very sensitive to graph connectivity and thus not a good preconditioning strategy in general. Preconditioning approach II, however, not only produces the best performance among all, but also shines with its robustness to graph topology



(a)



(b)

Figure 4.3: Performance comparison over a star graph (a) and a lollipop graph (b).

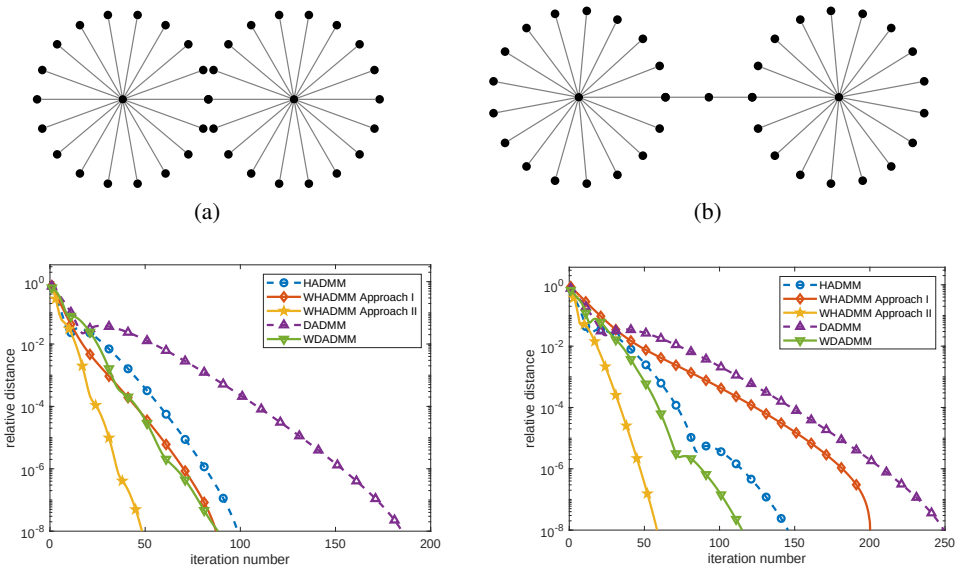


Figure 4.4: Results of performance comparison over graph (a) shown in (c), and graph (b) shown in (d).

changes, with a minimal performance difference.

4.6 Chapter summary

In this chapter, we provide the optimal design of edge weights for decentralized optimization using weighted hybrid ADMM which subsumes decentralized ADMM as special cases. The optimal weights are obtained via finding the best preconditioning matrix that yields the fastest convergence of ADMM. The developed approach fills the gap of existing results and works well for both star graphs and more general ones, with optimal guarantee. Numerical experiments show that preconditioned ADMM not only converges faster but also demonstrates robustness to topology changes.

4.7 Appendix

4.7.1 Derivation of Proposition 1

The ADMM iterations (3.3) become obvious upon obtaining the augmented Lagrangian of problem (4.3)

$$L(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \sum_{i}^{n} f_i(\boldsymbol{x}_i) + \lambda^{\mathsf{T}}(\hat{\boldsymbol{A}}\boldsymbol{x} - \hat{\boldsymbol{B}}\boldsymbol{z}) + \frac{\rho}{2}\|\hat{\boldsymbol{A}}\boldsymbol{x} - \hat{\boldsymbol{B}}\boldsymbol{z}\|_2^2.$$

To obtain per iteration updates, we minimize the augmented Lagrangian with respect to each variable

$$-\hat{\boldsymbol{A}}^{\mathsf{T}}\lambda^k - \gamma(\hat{\boldsymbol{A}}\boldsymbol{x}^{k+1} - \hat{\boldsymbol{B}}\boldsymbol{z}^k) \in \partial f(\boldsymbol{x}^{k+1}) \tag{4.19a}$$

$$-\hat{\boldsymbol{B}}^{\mathsf{T}}\lambda^k - \gamma\hat{\boldsymbol{B}}^{\mathsf{T}}(\hat{\boldsymbol{A}}\boldsymbol{x}^{k+1} - \hat{\boldsymbol{B}}\boldsymbol{z}^{k+1}) = 0 \tag{4.19b}$$

$$\lambda^{k+1} = \lambda^k + \gamma(\hat{\boldsymbol{A}}\boldsymbol{x}^{k+1} - \hat{\boldsymbol{B}}\boldsymbol{z}^{k+1}) \tag{4.19c}$$

where ∂f is the subdifferential of f. Left multiplying (4.19c) by $\hat{\boldsymbol{B}}^{\mathsf{T}}$ and adding it to (4.19b), we obtain

$$\hat{\boldsymbol{B}}^{\mathsf{T}}\lambda^{k+1} = 0, \quad k \geq 0. \tag{4.20}$$

Notice (4.20) holds only for $\lambda^k,\ k \geq 1$. With $\hat{\boldsymbol{B}}^{\mathsf{T}}\lambda^0 = \mathbf{0}$, it can be guaranteed that $\hat{\boldsymbol{B}}^{\mathsf{T}}\lambda^k = \mathbf{0}$ holds for all $k \geq 0$. Then we can obtain closed-form update for $\boldsymbol{z}$ by eliminating $\hat{\boldsymbol{B}}^{\mathsf{T}}\lambda^k$ and then left multiplying $\hat{\boldsymbol{B}}^{\mathsf{T}}$ to (4.19b)

$$\boldsymbol{z}^{k+1} = \hat{\boldsymbol{E}}^{-1}\hat{\boldsymbol{C}}^{\mathsf{T}}\boldsymbol{x}^{k+1}. \tag{4.21}$$

Left multiply (4.19c) by $\hat{\boldsymbol{A}}^{\mathsf{T}}$ and let $\boldsymbol{y} = \hat{\boldsymbol{A}}^{\mathsf{T}}\lambda$, then plug (4.21) into (4.19a) and (4.19c), we arrive at proposition 4.

4.8 Technical proofs in Section 4

To simplify the notation, all the lemmas and theorems in Section 4 are proved for unweighted hypergraphs. The results carry over to weighted ones since all the relations of important matrices remain the same.

4.8.1 Proof of Lemma 3

We first show that $\boldsymbol{G}$ is positive semidefinite (PSD). For any matrix $\boldsymbol{A}$, the product $\boldsymbol{A}\boldsymbol{A}^\mathsf{T}$ is always PSD since for any $\boldsymbol{x}$ of appropriate size, we have

$$\boldsymbol{x}^\mathsf{T}\boldsymbol{A}\boldsymbol{A}^\mathsf{T}\boldsymbol{x} = \|\boldsymbol{A}^\mathsf{T}\boldsymbol{x}\|_2^2 \geq 0.$$

Therefore, matrix $\boldsymbol{G} = \boldsymbol{A}\boldsymbol{A}^\mathsf{T} + \boldsymbol{B}\boldsymbol{B}^\mathsf{T}$ is the sum of two PSD matrices, which is also PSD.

The rank of $\boldsymbol{G}$ is not easy to find out. Instead, we introduce another matrix

$$\boldsymbol{G}' = \begin{bmatrix} \boldsymbol{D} & \boldsymbol{C} \\ \boldsymbol{C}^\mathsf{T} & \boldsymbol{E} \end{bmatrix}$$

where $\boldsymbol{C}, \boldsymbol{D}, \boldsymbol{E}$ are the important matrices related to the underlying communication graph, see Section 4.2 for details. Since $\boldsymbol{D} = \boldsymbol{A}^\mathsf{T}\boldsymbol{A}$, $\boldsymbol{E} = \boldsymbol{B}^\mathsf{T}\boldsymbol{B}$, and $\boldsymbol{C} = \boldsymbol{A}^\mathsf{T}\boldsymbol{B}$, we can decompose $\boldsymbol{G}'$ to the product of two matrices as $\boldsymbol{G}' = \boldsymbol{P}^\mathsf{T}\boldsymbol{P}$, where $\boldsymbol{P} = [\boldsymbol{A} \;\; \boldsymbol{B}] \in \mathbb{R}^{T\times(N+M)}$. Interestingly, matrix $\boldsymbol{G}$ can also be decomposed as $\boldsymbol{G} = \boldsymbol{P}\boldsymbol{P}^\mathsf{T}$. According to matrix theory, the nonzero eigenvalues of $\boldsymbol{G}$ and $\boldsymbol{G}'$ are the same. Thus, we have

$$\operatorname{rank}(\boldsymbol{G}) = \operatorname{rank}(\boldsymbol{G}').$$

Now we can investigate the rank of $\boldsymbol{G}'$ in order to find the rank of $\boldsymbol{G}$. We show that $\boldsymbol{G}'$ is congruent to a block diagonal matrix. Consider the decomposition

$$\boldsymbol{G}' = \begin{bmatrix} \boldsymbol{D} & \boldsymbol{C} \\ \boldsymbol{C}^\mathsf{T} & \boldsymbol{E} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{C}\boldsymbol{E}^{-1} \\ & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{D} - \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top & \\ & \boldsymbol{E} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \\ \boldsymbol{E}^{-1}\boldsymbol{C}^\top & \boldsymbol{I} \end{bmatrix} \tag{4.22}$$

Define

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{C}\boldsymbol{E}^{-1} \\ & \boldsymbol{I} \end{bmatrix}, \qquad \boldsymbol{\Lambda} = \begin{bmatrix} \boldsymbol{D} - \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\top & \\ & \boldsymbol{E} \end{bmatrix}$$

then $\boldsymbol{G}'$ is congruent to diagonal matrix $\boldsymbol{\Lambda}$, i.e., $\boldsymbol{G}' = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\mathsf{T}$. Since $\boldsymbol{Q}$ is nonsingular, we conclude that

$$\operatorname{rank}(\boldsymbol{G}') = \operatorname{rank}(\boldsymbol{\Lambda}) = \operatorname{rank}(\boldsymbol{D} - \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^\mathsf{T}) + \operatorname{rank}(\boldsymbol{E}). \tag{4.23}$$

The edge degree matrix $\boldsymbol{E}$ is diagonal and positive definite, thus

$$\operatorname{rank}(\boldsymbol{E}) = M.$$

The $\boldsymbol{L}_{\mathscr{G}} = \boldsymbol{D} - \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^{\mathsf{T}} \in \mathbb{R}^{N\times N}$ is a graph Laplacian of the hypergraph [7], which is a positive semidefinite matrix. Moreover, if the hypergraph is connected, then the multiplicity of zero eigenvalue is 1, which implies

$$\operatorname{rank}(\boldsymbol{D} - \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^{\mathsf{T}}) = N - 1.$$

Therefore, we have

$$\operatorname{rank}(\boldsymbol{G}) = \operatorname{rank}(\boldsymbol{G}') = \operatorname{rank}(\boldsymbol{D} - \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^{\mathsf{T}}) + \operatorname{rank}(\boldsymbol{E}) = N + M - 1.$$

4.8.2 Proof of theorem 4

Lemma 9. *Consider a positive definite matrix $\boldsymbol{M}$. Then under congruent transformations $\boldsymbol{M}$ remains positive definite, i.e., $\boldsymbol{P}^{\mathsf{T}}\boldsymbol{M}\boldsymbol{P} \succeq 0$, for any invertible matrix $\boldsymbol{P}$.*

lemma 9 implies that we can rearrange some block structured matrix without changing its positive definiteness. For example, consider a matrix consisting of four blocks. It holds that

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} \succeq 0 \Longleftrightarrow \begin{bmatrix} \boldsymbol{D} & \boldsymbol{C} \\ \boldsymbol{B} & \boldsymbol{A} \end{bmatrix} \succeq 0.$$

Now consider the block structured matrix

$$\boldsymbol{G}' = \begin{bmatrix} \boldsymbol{D} & \boldsymbol{C} \\ \boldsymbol{C}^{\mathsf{T}} & \boldsymbol{E} \end{bmatrix} = \boldsymbol{P}^{\mathsf{T}}\boldsymbol{P}$$

where $\boldsymbol{C}, \boldsymbol{D}, \boldsymbol{E}$ are the incidence, node and edge degree matrix, respectively (see Section 2), and $\boldsymbol{P} = [\boldsymbol{A} \;\; \boldsymbol{B}]$. We also have $\boldsymbol{G} = \boldsymbol{A}\boldsymbol{A}^{\mathsf{T}} + \boldsymbol{B}\boldsymbol{B}^{\mathsf{T}} = \boldsymbol{P}\boldsymbol{P}^{\mathsf{T}}$, which implies $\boldsymbol{G}$ is also positive semidefinite (PSD).

The graph Laplacian, $\boldsymbol{L}_{\mathscr{G}} = \boldsymbol{D} - \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^{\mathsf{T}}$, is the Schur complement of $\boldsymbol{E}$, i.e., $\boldsymbol{G}'/\boldsymbol{E} = \boldsymbol{D} - \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^{\mathsf{T}} = \boldsymbol{L}_{\mathscr{G}}$. According to the interlacing property of Schur compliment [119,

Corollary 2.3] [1], we have

$$\lambda_i(\boldsymbol{G}') \geq \lambda_i(\boldsymbol{G}'/\boldsymbol{E}) \geq \lambda_{i+M}(\boldsymbol{G}')$$

for $i = 1, 2, \ldots, N$. Since $\boldsymbol{G}$ and $\boldsymbol{G}'$ share nonzero eigenvalues, we arrive at

$$\lambda_i(\boldsymbol{G}) \geq \lambda_i(\boldsymbol{L}_{\mathscr{G}}) \geq \lambda_{i+M}(\boldsymbol{G}), \quad i = 1, 2, \ldots, N-1.$$

4.8.3 Proof of Corollary 1

For a connected hypergraph, the graph Laplacian $\boldsymbol{L}_{\mathscr{G}} = \boldsymbol{D} - \boldsymbol{C}\boldsymbol{E}^{-1}\boldsymbol{C}^{\mathsf{T}}$ has a zero eigenvalue with multiplicity one. Thus, we have

$$\begin{aligned} \lambda_{\max}(\boldsymbol{L}_{\mathscr{G}}) &= \lambda_1(\boldsymbol{L}_{\mathscr{G}}) \\ \lambda_{\min>0}(\boldsymbol{L}_{\mathscr{G}}) &= \lambda_{N-1}(\boldsymbol{L}_{\mathscr{G}}). \end{aligned} \tag{4.24}$$

According to Lemma 3, $\boldsymbol{G}' \in \mathbb{R}^{(N+M)\times(N+M)}$ has a rank $N+M-1$, thus $\boldsymbol{G}'$ is singular. As a result, we obtain

$$\begin{aligned} \lambda_{\max}(\boldsymbol{G}) &= \lambda_{\max}(\boldsymbol{G}') = \lambda_1(\boldsymbol{G}), \\ \lambda_{\min>0}(\boldsymbol{G}) &= \lambda_{\min>0}(\boldsymbol{G}') = \lambda_{M+N-1}(\boldsymbol{G}). \end{aligned} \tag{4.25}$$

Theorem 1 shows that

$$\begin{aligned} \lambda_{\max}(\boldsymbol{G}) = \lambda_1(\boldsymbol{G}) \geq \lambda_1(\boldsymbol{L}_{\mathscr{G}}) = \lambda_{\max}(\boldsymbol{L}_{\mathscr{G}}) \\ \lambda_{\min>0}(\boldsymbol{L}_{\mathscr{G}}) = \lambda_{N-1}(\boldsymbol{L}_{\mathscr{G}}) \geq \lambda_{M+N-1}(\boldsymbol{L}_{\mathscr{G}}) = \lambda_{\min>0}(\boldsymbol{G}). \end{aligned}$$

Combined with $\lambda_1(\boldsymbol{L}_{\mathscr{G}}) \geq \lambda_2(\boldsymbol{L}_{\mathscr{G}}) \geq \ldots \geq \lambda_{N-1}(\boldsymbol{L}_{\mathscr{G}})$, we have

$$\lambda_{\max}(\boldsymbol{G}) \geq \lambda_{\max}(\boldsymbol{L}_{\mathscr{G}}) \ldots \geq \lambda_{\min>0}(\boldsymbol{L}_{\mathscr{G}}) \geq \lambda_{\min>0}(\boldsymbol{G}).$$

[1] F. Zhang, Ed., *The Schur Complement and Its Applications*. Boston, MA: Springer Science & Business Media, 2005, vol. 4.

Chapter 5

Fast Asynchronous Decentralized Optimization: Allowing Multiple Masters

5.1 Introduction

Consider the following distributed optimization problem over N networked computing nodes (henceforth called workers) with node-specific cost functions and privately available data

$$\min_{\boldsymbol{x}} \quad \sum_{i=1}^{N} f_i(\boldsymbol{x}) + h(\boldsymbol{x}) \tag{5.1}$$

where $\boldsymbol{x} \in \mathbb{R}^p$ is the global decision variable, f_i denotes the local cost function at node i and h a (not necessarily smooth) regularizer. The goal is to find the optimal solution by cooperatively solving the per-worker subproblems. This setting arises frequently in estimation, learning and control tasks [37, 9, 89, 5, 96]. Among various solvers, alternating direction method of multiplier (ADMM) has gained popularity for its decomposability and flexibility [5, 9]. Typically, ADMM-based solvers come in two formats: i) *centralized*, where a single master node is connected all workers [9]; and ii) *decentralized*, where no master is present and workers talk to single-hop neighbors only [37, 103]. Until recently [71], there has been no principled approaches to dealing with multiple masters. However, [71] dealt with *synchronous* algorithms

that require global coordination, thus challenging their implementation.

Asynchronous algorithms may be more appealing in some settings, especially for decentralized optimization, since they do not need global coordination and consequently are more efficient when workers differ significantly in computing speed [91, 114, 115, 68, 61, 1, 2]. With prior art dealing with either *centralized* or *decentralized* operations, the main contribution of this work to develop an ADMM-based asynchronous decentralized solver of (5.1) using multiple masters is well motivated.

Related work. From the plethora of distributed optimization schemes, we will focus on ADMM-based ones, which can be split in two categories: synchronous and asynchronous.

Synchronous distributed optimization has been studied extensively for decades; see e.g., [5]. These methods may be *centralized* [9] or *decentralized* [37, 103, 79], depending on whether a master (fusion center) is present or not. This setup of multiple masters remains a largely uncharted territory. Progress was made recently in [54] where a cluster of workers are handled together, but no explicit means to accommodate multiple masters. A novel synchronous approach that is capable of handling multiple masters was proposed in [71].

Similarly, asynchronous algorithms are either *centralized* or *decentralized*. Centralized ones are popular, and are easier to analyze and implement [120, 16, 49, 91, 26, 48, 81], but the single master operation faces single-point failure and bottleneck related challenges that limit the overall system performance. Consequently, decentralized alternatives have been developed [114, 91, 88, 63, 94]. But no method is available to accommodate multiple masters except for the asynchronous version of [53].

Contributions. We classify our contributions as follows: C1) we develop AH-ADMM that is able to accommodate multiple masters; C2) we show that AH-ADMM enables topology-aware acceleration of decentralized algorithms without changing the underlying network; and C3) we establish convergence of AH-ADMM for nonconvex functions.

The rest of the chapter is organized as follows. Sec. 5.2 presents detailed derivation of AH-ADMM, Sec 5.3 deals with topology-aware acceleration, Sec. 5.4 presents convergence analysis, Sec. 5.5 shows numerical tests and Sec. 5.6 concludes this chapter.

5.2 Asynchronous Hybrid ADMM

This section provides background about optimization with multiple masters, followed by development of AH-ADMM.

5.2.1 Accommodating multiple masters

The presence of multiple masters makes communication among workers much more complicated. Some workers may be connected to masters and also other worker, thus necessitating exchange of information to both. We refer to such communication constraints as *hybrid constraints*.

Communication constraints can be effectively described by graphs. The centralized case corresponds to a star graph, with the master at the center and workers around. The decentralized case can be described by a connected graph, whose nodes corresponds to workers and edges represent communication between neighbors. Hybrid constraints are best depicted by *hypergraphs*. Each master is described by a hyperedge consisting of all its connected workers. Fig. 5.1 is an example of hybrid constraints with a master connected to workers 1, 2 and 3.

5.2.2 Problem formulation

A hypergraph is a tuple $\mathcal{H} := (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \dots, N\}$ is the vertex set, and $\mathcal{E} := \{\mathcal{E}_i | \mathcal{E}_i \subset \mathcal{V}\}$ denotes the set of hyperedges. Let N be the number of nodes, and $M = |\mathcal{E}|$ the number of (hyper)edges. A vertex i and an edge $\mathcal{E}_j$ are said to be *incident* if $i \in \mathcal{E}_j$. A simple edge can be seen as an hyperedge consisting of two nodes.

By creating copies $\boldsymbol{x}_i$ at each node and assigning each hyperedge $\mathcal{E}_i$ an auxiliary variable $\boldsymbol{z}_i$, we can write hybrid constraints uniformly as $\boldsymbol{x}_i = \boldsymbol{z}_j,\ \forall i \in \mathcal{E}_j$. Let T be the number of constraints. Consider now vectors $\boldsymbol{x} \in \mathbb{R}^{Np}$, $\boldsymbol{z} \in \mathbb{R}^{Mp}$ concatenating $\{\boldsymbol{x}_i\}, \{\boldsymbol{z}_j\}$, and also matrices $\tilde{\boldsymbol{A}} \in \mathbb{R}^{T \times N}$, $\tilde{\boldsymbol{B}} \in \mathbb{R}^{T \times M}$ whose t-th row $\tilde{A}_{ti} = 1$, $\tilde{B}_{tj} = 1$ corresponds to t-th constraint $\boldsymbol{x}_i = \boldsymbol{z}_j,\ i \in \mathcal{E}_j$. Upon defining $\boldsymbol{A} = \tilde{\boldsymbol{A}} \otimes \boldsymbol{I}_p$, $\boldsymbol{B} = \tilde{\boldsymbol{B}} \otimes \boldsymbol{I}_p$, where $\otimes$ is the Kronecker product, problem (5.1) can be formulated as

$$\begin{aligned} \min_{\boldsymbol{x}_i} \ & \sum_{i=1}^{N} f_i(\boldsymbol{x}_i) + \sum_{j=1}^{M} h_j(\boldsymbol{z}) \\ \text{s. to } & \boldsymbol{A}\boldsymbol{x} - \boldsymbol{B}\boldsymbol{z} = \boldsymbol{0} \end{aligned} \tag{5.2}$$

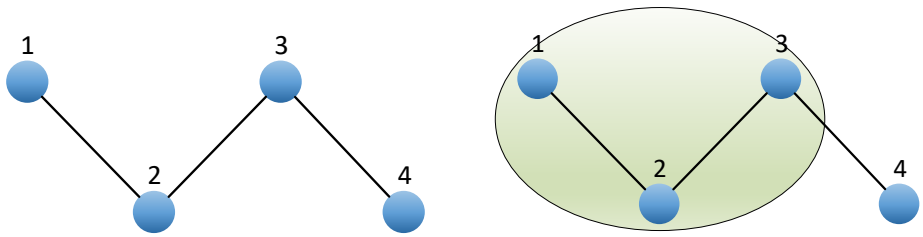


Figure 5.1: An example of hybrid constraints described by a hypergraph. (a) is the underlying graph, and (b) is a hypergraph where the shaded ellipsoid denotes an hyperedge.

where $h_j := h/M$. Let $\tilde{\boldsymbol{C}} \in \mathbb{R}^{N\times M}$ be the signless incidence matrix of the hypergraph, meaning $\tilde{C}_{ij} = 1$ if node i and edge j are incident, and $\tilde{C}_{ij} = 0$ otherwise. Let D_i denote the degree of node i (number of incident edges), E_j the degree of hyperedge j (number of incident nodes), and diagonal matrix $\tilde{\boldsymbol{D}} \in \mathbb{R}^{N\times N}$ and $\tilde{\boldsymbol{E}} \in \mathbb{R}^{M\times M}$ collecting $\{D_i\}_{i=1}^N$ and $\{E_j\}_{j=1}^M$, respectively. With $\boldsymbol{C} := \tilde{\boldsymbol{C}} \otimes \boldsymbol{I}_p$, $\boldsymbol{D} := \tilde{\boldsymbol{D}} \otimes \boldsymbol{I}_p$, one can show that $\boldsymbol{A}^\top \boldsymbol{A} = \boldsymbol{D}$, $\boldsymbol{B}^\top \boldsymbol{B} = \boldsymbol{E}$ and $\boldsymbol{A}^\top \boldsymbol{B} = \boldsymbol{C}$ (see [71] for the proof).

Example In Fig. 5.1, we have $N = 4$, $M = 2$, and $T = 5$. Specifically, the constraints are $x_i = z_1, i = 1, 2, 3$; $x_3 = z_2$, $x_4 = z_2$. These are expressible in compact form as $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{B}\boldsymbol{z}$, where $\tilde{\boldsymbol{A}}$ and $\tilde{\boldsymbol{B}}$ are given by

$$\tilde{\boldsymbol{A}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad \tilde{\boldsymbol{B}} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}. \tag{5.3}$$

5.2.3 Asynchronous Hybrid ADMM

The setup of AH-ADMM is similar to that of synchronous hybrid ADMM [71], except that the former has to cope with asynchrony. The high-level description of AH-ADMM is as follows. i) Each worker solves its subproblem individually and communicates the solution to *incident* master(s), then starts waiting until responses from its masters arrive; ii) each master updates its solution whenever updates from a preselected number of workers have been received. Received values can be outdated; and iii) after updating, each master sends results to all *received* workers

it received updates from, in order for them to update their associated dual variables.

To describe the process mathematically, we first introduce some definitions. Let k denote a *virtual* iteration counter that is increased by 1 when *any* master finishes its update. By definition, at iteration k, there is exactly *one* master updating, denoted by j_k. Let $\mathcal{A}^k$ be the active set at iteration k, containing received workers of master j_k.

The updates of AH-ADMM are obtained by optimizing the augmented Lagrangian with respect to corresponding variables (see [71] for details). However, due to asynchrony, $\boldsymbol{x}_i^{k+1}$ may depend on outdated values $\hat{\boldsymbol{z}}_j$ as incident masters could have updated again while worker i is computing. On the other hand, the update of $\boldsymbol{z}_{j_k}^{k+1}$ always has access to latest values of $\{\boldsymbol{x}_i^{k+1}, i \in \mathcal{A}^k\}$, thanks to the updating order, and likewise for $\boldsymbol{\lambda}_t^{k+1}$. All other values not involved remain the same. Equivalently, updates of active workers $i \in \mathcal{A}^k$ can be seen as occuring right before the update of master j_k. Specifically, the updates are

$$\boldsymbol{x}_i^{k+1} = \arg\min_{\boldsymbol{x}} f_i(\boldsymbol{x}_i) + \boldsymbol{x}_i^\top \boldsymbol{u}_i^k + \frac{\rho}{2}\Big(D_i\|\boldsymbol{x}_i\|^2 - 2\boldsymbol{x}_i^\top \sum_{i\sim j} \hat{\boldsymbol{z}}_j\Big), i \in \mathcal{A}^k \tag{5.4a}$$

$$\boldsymbol{z}_{j_k}^{k+1} = \arg\min_{\boldsymbol{z}_{j_k}} h_{j_k}(\boldsymbol{z}_{j_k}) - \boldsymbol{z}_{j_k}^\top \boldsymbol{v}_{j_k}^k + \frac{\gamma}{2}\|\boldsymbol{z}_{j_k} - \boldsymbol{z}_{j_k}^k\|^2 + \frac{\rho}{2}\Big(E_{j_k}\|\boldsymbol{z}_{j_k}\|^2 - 2\boldsymbol{z}_{j_k}^\top \sum_{i\in\mathcal{A}^k} \boldsymbol{x}_i^{k+1}\Big) \tag{5.4b}$$

$$\boldsymbol{\lambda}_t^{k+1} = \boldsymbol{\lambda}_t^k + \rho(\boldsymbol{A}\boldsymbol{x}_i^{k+1} - \boldsymbol{B}\boldsymbol{z}_j^{k+1}),\ t = \mathcal{T}(i,j) \tag{5.4c}$$

where $\mathcal{T}(i,j) = t$ describes the mapping of worker i and master j to the associated multiplier $\boldsymbol{\lambda}_t$, while $\boldsymbol{u} := [\boldsymbol{u}_1^\top, \ldots, \boldsymbol{u}_N^\top]^\top = \boldsymbol{A}^\top\boldsymbol{\lambda}$ and $\boldsymbol{v} := [\boldsymbol{v}_1^\top, \ldots, \boldsymbol{v}_M^\top]^\top = \boldsymbol{B}^\top\boldsymbol{\lambda}$ are change of variables. We include a proximal term in (5.4b) to guarantee the convergence of AH-ADMM, see Theorem 5. The AH-ADMM algorithm is better understood by considering masters and workers separately, see Algorithms 3 and 4.

5.3 Topology-aware Acceleration

Hybrid ADMM generally converges faster than its decentralized counterparts [71], which is not surprising due to the presence of multiple masters. When no masters are available, AH-ADMM reduces to AD-ADMM, no longer providing any performance gain. Therefore, we are more interested in the question "*Can we benefit from AH-ADMM even if no master exists?*" The answer is *yes*. The idea is to create *virtual masters* inside workers and employ AH-ADMM

Algorithm 3: AH-ADMM: worker side

Input: ρ, $\boldsymbol{\lambda}^0, \boldsymbol{z}^0$
while *stop criterion not satisfied* **do**
  for *worker i = 1,2,..., N* **do**
    update $\boldsymbol{x}_i$ by (5.4a)
    while *not enough masters are received* **do**
      wait
    update $\boldsymbol{\lambda}_t$ by (5.4c)
    send $\{\boldsymbol{x}_i, \boldsymbol{\lambda}_t\}$ to incident masters and neighboring workers

Algorithm 4: AH-ADMM: master side

Input: ρ, $\boldsymbol{\lambda}^0, \boldsymbol{x}^0, \boldsymbol{z}^0$
while *stop criterion not satisfied* **do**
  for *master j = 1,2,..., M* **do**
    while *not enough workers are received* **do**
      wait
    update $\boldsymbol{z}_j$ by (5.4b)
    send $\boldsymbol{z}_j$ to all received workers

afterwards. This technique transforms a decentralized optimization problem to one that can be tackled using hybrid ADMM without physically adding nodes or edges.

The first step to apply this technique is to select workers as hosts that serves as *virtual masters* inside. Subsequently, we connect each virtual master to all neighbors of its host and the host itself, which can be done using all edges of the host. Repeating this procedure creates multiple masters, with the help of whom it becomes possible to employ AH-ADMM subsequently; see also Fig. 5.2 for an example. Notice that in the process, no *physical* nodes or edges have been deployed since virtual masters are just *logical* entities. However, host nodal updates increase complexity. AH-ADMM is also flexible to allow the deployment of any number of virtual masters, a balanced means of boosting performance.

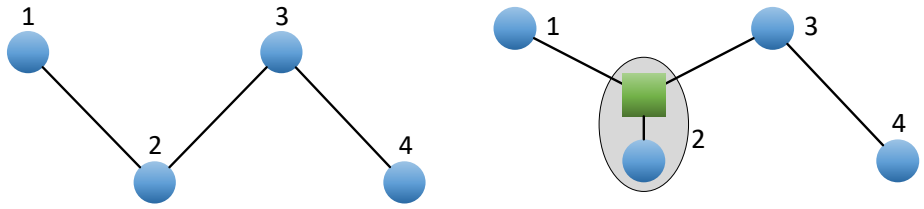


Figure 5.2: Illustration of topology-aware acceleration. The underlying graph is (a), and node 2 is selected as host to serve as virtual master, depicted by the square. The shaded ellipsoid in (b) plays the same role as node 2 in (a).

5.4 Convergence Analysis

In this section, we analyze the convergence of AH-ADMM. Let τ be the maximum delay, which means that every worker performs update at least once during τ iterations; and $F^\star$ the optimal objective value of (5.2).

Inspired by [14], the following theorem establishes the convergence of AH-ADMM. The analysis in [14] assumes a single master. It is nontrivial to extend the reasoning in [14] to multiple masters, because the dual variables in $\boldsymbol{\lambda}$ are coupled.

Theorem 5. *Suppose that the maximum delay is finite $\tau < \infty$, f_i is twice differentiable and its gradient ∇f_i is Lipschitz with constant L, while h is proper and convex. Then sequences $\{\boldsymbol{x}_i^k\}_{i=1}^N$, $\{\boldsymbol{z}_j^k\}_{j=1}^M$ and $\{\boldsymbol{\lambda}_t^k\}_{t=1}^T$ converge to some limit points satisfying the KKT condition of problem* (5.2)*, provided that*

$$0 \leq L_\rho(\boldsymbol{x}^0, \boldsymbol{z}^0, \boldsymbol{\lambda}^0) - F^\star < \infty \tag{5.5}$$

$$\rho > \frac{D_{min} + L + \sqrt{(D_{min} + L)^2 + 8L^2 D_{min}}}{2D_{min}} \tag{5.6}$$

$$\gamma \geq \frac{S_m(\tau - 1)^2\rho^2 - \rho E_{min}}{2} \tag{5.7}$$

where $S_m = \max_k |\mathcal{A}^k|$, $D_{min} = \min_i D_i$, $E_{min} = \min_j E_j$.

Theorem 5 shows that the solution given by AH-ADMM is guaranteed to converge to some KKT points of (5.2), as long as ρ and γ are large enough. Note that f_i does not need to be convex. Different from [114, 91], Theorem 5 does not need assumptions of random activation of workers, thus being able to cope also with deterministic settings.

Theorem 5 implies that ρ and γ should be sufficiently large to guarantee the convergence of AH-ADMM. Speicifically, (5.7) suggests that a large γ is needed when the maximum delay is large. Also, (5.6) implies that ρ can be smaller when the cost functions are smoother.

5.5 Numerical Experiments

In this section, we carry out numerical experiments to test the acceleration merits of AH-ADMM, and compare with asynchronous decentralized ADMM (AD-ADMM) [114, 91], synchronous decentralized ADMM (SD-ADMM) [37], and synchronous hybrid ADMM [71].

Different from existing works [91, 115, 114], our method does not assume every worker can activate each iteration. For this reason, it can deal with more general settings. For example, each worker has a positive activation probability in [91, 115] at each iteration, while one edge is randomly selected each time in [114]. Both assumptions exclude the case of deterministic activation patterns, as verified in the following experiment.

In our experiment, the speed of each worker is deterministic and initialized randomly by drawing from a uniform distribution $U[1, 10]$, so that the fastest worker can be 10 times faster than the slowest one, which also ensures bounded delays. We evaluate the performance by plotting its relative error $\frac{\|\boldsymbol{x}^k-\boldsymbol{x}^\star\|}{\|\boldsymbol{x}^0-\boldsymbol{x}^\star\|}$ against wall clock time. The optimal solution $\boldsymbol{x}^\star$ can either be computed directly if it admits a closed-form solution, or be obtained using CVX [44, 45]. We also show average working and waiting time of workers to demonstrate the effects of the threshold of masters.

The decentralized *sparse compressed sensing* we tested aims at reconstructing a sparse unknown vector $\boldsymbol{x} \in \mathbb{R}^p$ through nodal measurements $\boldsymbol{b}_i = \boldsymbol{H}_i\boldsymbol{x} + \boldsymbol{e}_i,\ i = 1, \ldots, N$, where $\boldsymbol{H}_i \in \mathbb{R}^{n_i \times p}$ is the sensing matrix of node i and $\boldsymbol{e}_i$ represents a vector of i.i.d. Gaussian noise. When $p > \sum_{i=1}^N n_i$, there are more unknowns than measurements. The sparsity of $\boldsymbol{x}$ suggests solving a L_1-regularized least-squares problem

$$\min_{\boldsymbol{x}} \quad \sum_{i=1}^N \frac{1}{2}\|\boldsymbol{H}_i\boldsymbol{x} - \boldsymbol{b}_i\|^2 + \mu\|\boldsymbol{x}\|_1 \tag{5.8}$$

where $\mu > 0$ is the regularization parameter.

We consider a ring graph of 10 nodes, and set $n_i = 3$ and $p = 40$ such that $p > \sum_{i=1}^N n_i$. The entries of $\boldsymbol{H}_i$ are generated from the standard Gaussian distribution $N(0, 1)$, and then

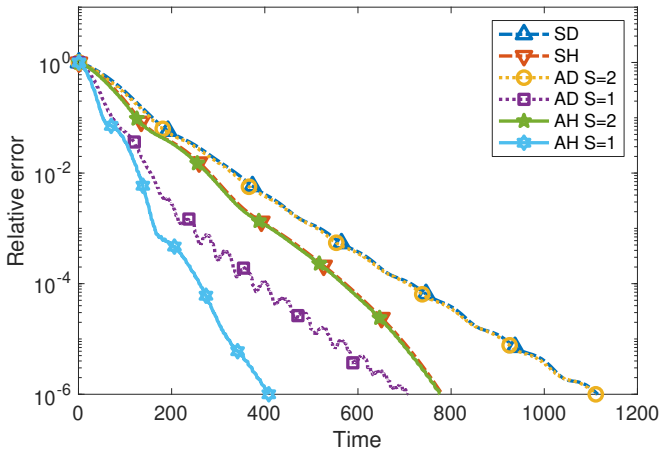


Figure 5.3: Relative error of SD-ADMM (SD), SH-ADMM (SH), AD-ADMM (AD) and AH-ADMM (AH) with different threshold (S) vs. wall clock time.

normalized so that $\|\boldsymbol{H}_i\|_2 = 1$. The unknown vector $\boldsymbol{x}$ is drawn from $N(0,1)$ with 10% nonzero entries, and subsequently $\boldsymbol{b}_i$ is generated. Since (5.8) admits no closed-form solution, we solve it using CVX to obtain the optimal solution $\boldsymbol{x}^\star$. We set $\gamma = 0$ and tune ρ to be nearly optimal.

Fig. 5.3 depicts the relative error against wall clock time. When $S = 2$, AH-ADMM and AD-ADMM are almost equivalent to their synchronous counterparts, while AD-ADMM with $S = 1$ corresponds to the case that each node updates as soon as it receives information from any neighbor, thus eliminating waiting time. One can immediately make two observations: a) asynchronous approaches converge at least as fast as, if not faster, than their decentralized counterparts; and b) hybrid approaches always outperform their decentralized counterparts, showcasing their promising potential.

Fig. 5.4 compares average working and waiting time of different approaches. Again, we notice that a) asynchronous approaches reduce or even eliminate waiting time, thus improving efficiency; and b) hybrid approaches consume significantly less working and waiting.

5.6 Chapter summary

This chapter presents an asynchronous distributed optimization algorithm, capable of handling multiple masters, AH-ADMM, which not only broadens the applicability of ADMM, but also

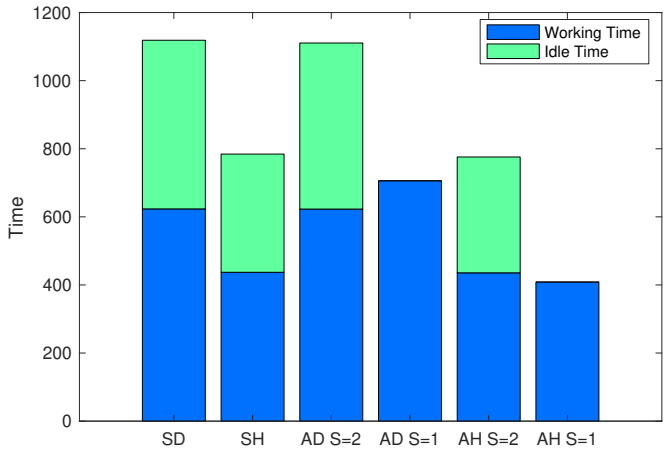


Figure 5.4: Average working and waiting time of workers of SD-ADMM, SH-ADMM, AD-ADMM and AH-ADMM.

yields a technique that significantly accelerates the convergence of decentralized ADMM without changing the underlying topology. A convergence result is provided and numerical experiments are performed to compare AH-ADMM against decentralized approaches.

5.7 Appendix

Suppose there are N workers and M masters in total, and we denote $x_i \in \mathbb{R}^l$ the decision variable associated with worker i, and $z_j \in \mathbb{R}^l$ the variable associated with master j. For the sake of simplicity and clarity, all subsequent proofs would be based on the case $l = 1$, and it is trivial to generalize to the cases $l >= 2$.

For an arbitrary labeling order, we use $\{1, 2, \ldots, N\}$ and $\{1, 2, \ldots, M\}$ to refer to a specific worker and master. Let $\boldsymbol{D}$ ($\boldsymbol{E}$) be the diagonal degree matrix of workers (masters) with the i-th (j-th) diagonal element D_i (E_j) referring to the degree of worker i (master j). Let S be the minimum number of workers that each master must receive before continuing; also let τ be the maximum delay in one iteration. Denotes $\mathcal{A}_j^k$ the active set of workers that have been received by master j at iteration k.

The asynchronism of AH-ADMM algorithm imposes great challenge for proof since the state of workers and masters can be complicated. To logically represent the state of algorithm, we use iteration index k to indicate the number of updates happened on *all masters* from the

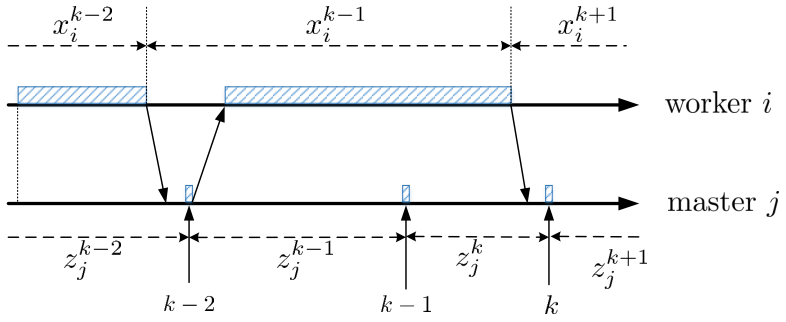


Figure 5.5: Timeline of worker and master updates

beginning until now. In other words, global counter k would be increased by one each time one master finishes one updates. This counting scheme is a natural choice considering each master needs to wait for at least S workers before performing its own update.

A possible update sequence is illustrated in Figure 5.5, where the solid black arrows represent the clock time for worker i and master j, and shaded rectangles indicate the corresponding worker or master is in the process of generating a new update. The state of a worker or master stays the same without updates, even though the index counter k has increased due to others' activity. Notice that the update process of masters only last for a negligible period of time, since most of the weight lifting job is done by workers.

Suppose at iteration k, master j will update its value, then this process can be equivalently represented by

$$x_i^{k+1} := \arg\min_{x_i} f_i(x_i) + u_i^k x_i + \frac{\rho}{2}\left(D_i x_i^2 - 2x_i \sum_{i\sim j} \hat{z}_j^k\right),\ i \in \mathcal{A}_j^k \tag{5.9a}$$

$$z_j^{k+1} := \arg\min_{z_j} h_j(z_j) - v_j^k z_j + \frac{\rho}{2}\left(E_j z_j^2 - 2z_j \sum_{i\in\mathcal{A}_j^k} x_i^{k+1}\right) + \frac{\gamma}{2}\|z_j^{k+1} - z_j^k\|^2 \tag{5.9b}$$

$$\lambda_t^{k+1} := \lambda_t^{k+1} + \rho(x_i^{k+1} - z_j^{k+1}), \quad \mathcal{T}(i,j) = t \tag{5.9c}$$

where $i \sim j$ denotes worker i and master j are connected, and $\mathcal{T}$ is a mapping from (i,j) to t such that if worker i and master j are connected, i.e., $C_{ij} = 1$, then $A_{ti} = 1$ and $B_{tj} = 1$. For all other workers and masters, their values keep the same.

Define the Lagrangian as

$$L(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\lambda}) := \sum_{i=1}^{N} f_i(x_i) + \sum_{j=1}^{M} h_j(z_j) + \boldsymbol{\lambda}^\top (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{B}\boldsymbol{z}) + \frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{B}\boldsymbol{z}\|^2 \tag{5.10}$$

where $\boldsymbol{x} \in \mathbb{R}^N$, $\boldsymbol{z} \in \mathbb{R}^M$, $\boldsymbol{\lambda} \in \mathbb{R}^T$, $\boldsymbol{u} = \boldsymbol{A}^\top \boldsymbol{\lambda} \in \mathbb{R}^N$ and $\boldsymbol{v} = \boldsymbol{B}^\top \boldsymbol{\lambda} \in \mathbb{R}^N$. We need the following lemmas to prove Theorem 5.

Assumption 5. Each function f_i is twice differentiable and its gradient ∇f_i is Lipschitz continuous with a constant $L > 0$. The function h is proper and convex. Problem (5.2) is bounded below, i.e., $F^\star > -\infty$ where $F^\star$ is the optimal value.

Lemma 10. *Suppose Assumption 5 holds and $\rho > L$. Then it holds that*

$$\begin{aligned} & L(\boldsymbol{x}^{k+1}, \boldsymbol{z}^{k+1}, \boldsymbol{\lambda}^{k+1}) - L(\boldsymbol{x}^k, \boldsymbol{z}^k, \boldsymbol{\lambda}^k) \\ \leq & -\frac{\rho E_j + 2\gamma}{2}\|z_j^k - z_j^{k+1}\|^2 + \frac{1}{\rho}\|\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k\|^2 \\ & + \frac{\rho^2}{2}\sum_{i \in \mathcal{A}_j^k}\sum_{j \sim i}\|\hat{z}_j^k - z_j^k\|^2 + \sum_{i \in \mathcal{A}_j^k}\frac{D_i - \rho D_i}{2}\|x_i^{k+1} - x_i^k\|^2. \end{aligned} \tag{5.11}$$

Lemma 11. *Suppose that Assumption 5 holds and assume that $|\mathcal{A}_k| < S$ for all k and some constant $S \in [1, N]$. Then it holds that*

$$\sum_{k=0}^{K}\sum_{i \in \mathcal{A}_j^k}\sum_{j \sim i}\|\hat{z}_j^k - z_j^k\|^2 \leq S(\tau - 1)^2 \sum_{k=0}^{K}\|z_{j_k}^k - z_{j_k}^{k+1}\|^2. \tag{5.12}$$

5.8 Proof of Lemma 1

We need to bound the difference of Lagrangian for each iteration, then the difference between initial and limit value is readily available. Towards this end, we have

$$\begin{aligned} & L(\boldsymbol{x}^{k+1}, \boldsymbol{z}^{k+1}, \boldsymbol{\lambda}^{k+1}) - L(\boldsymbol{x}^k, \boldsymbol{z}^k, \boldsymbol{\lambda}^k) = \\ & \quad L(\boldsymbol{x}^{k+1}, \boldsymbol{z}^k, \boldsymbol{\lambda}^k) - L(\boldsymbol{x}^k, \boldsymbol{z}^k, \boldsymbol{\lambda}^k) + L(\boldsymbol{x}^{k+1}, \boldsymbol{z}^{k+1}, \boldsymbol{\lambda}^k) - L(\boldsymbol{x}^{k+1}, \boldsymbol{z}^k, \boldsymbol{\lambda}^k) \\ & \qquad + L(\boldsymbol{x}^{k+1}, \boldsymbol{z}^{k+1}, \boldsymbol{\lambda}^{k+1}) - L(\boldsymbol{x}^{k+1}, \boldsymbol{z}^{k+1}, \boldsymbol{\lambda}^k). \end{aligned} \tag{5.13}$$

Notice the Lagrangian (5.10) is separable across x_i, thus we can obtain the Lagrangian for each x_i

$$\begin{aligned} L_i(x_i^{k+1}, \boldsymbol{z}^k, \boldsymbol{\lambda}^k) &= f_i(x_i^{k+1}) + u_i^k x_i^{k+1} + \frac{\rho}{2}(D_i\|x_i^{k+1}\|^2 - 2x_i^{k+1}\sum_{j\sim i} z_j^k) \\ &= f_i(x_i^{k+1}) + u_i^k x_i^{k+1} + \frac{\rho}{2}(D_i\|x_i^{k+1}\|^2 - 2x_i^{k+1}\sum_{j\sim i} \hat{z}_j^k) \\ &\quad + \rho x_i \sum_{j\sim i}(\hat{z}_j^k - z_j^k). \end{aligned} \tag{5.14}$$

According to the optimality condition of x_i^{k+1}, we immediately know that

$$\nabla_{x_i} L_i(x_i^{k+1}, \boldsymbol{z}^{k+1}, \boldsymbol{\lambda}^{k+1}) = \rho \sum_{j\sim i}(\hat{z}_j^k - z_j^k). \tag{5.15}$$

The convexity of $f_i(\cdot)$ implies $L(x_i, \boldsymbol{z}, \boldsymbol{\lambda})$ is strongly convex with modulus $D_i\rho/2$. Therefore, by the strong convexity we have

$$\begin{aligned} &L_i(x_i^k, \boldsymbol{z}^k, \boldsymbol{\lambda}^k) - L_i(x_i^{k+1}, \boldsymbol{z}^k, \boldsymbol{\lambda}^k) \geq \\ &\qquad (\nabla_{x_i} L_i(x_i^{k+1}, \boldsymbol{z}^k, \boldsymbol{\lambda}^k))^\top (x_i^k - x_i^{k+1}) + \frac{\rho}{2} D_i \|x_i^k - x_i^{k+1}\|^2 \\ &\qquad = \rho \sum_{j\sim i} (\hat{z}_j^k - z_j^k)^\top (x_i^k - x_i^{k+1}) + \frac{\rho}{2} D_i \|x_i^k - x_i^{k+1}\|^2. \end{aligned} \tag{5.16}$$

Recall Young's inequality

$$\boldsymbol{a}^\top \boldsymbol{b} \leq \frac{1}{2c}\|\boldsymbol{a}\|^2 + \frac{c}{2}\|\boldsymbol{b}\|^2. \tag{5.17}$$

Let $c = 1/\rho$ and sum over i, we obtain

$$L(\boldsymbol{x}^{k+1}, \boldsymbol{z}^k, \boldsymbol{\lambda}^k) - L(\boldsymbol{x}^k, \boldsymbol{z}^k, \boldsymbol{\lambda}^k) \leq \sum_{i=1}^N \frac{D_i - \rho D_i}{2}\|x_i^k - x_i^{k+1}\|^2 - \frac{\rho^2}{2}\sum_{i=1}^N \sum_{j\sim i} \|\hat{z}_j^k - z_j^k\|^2. \tag{5.18}$$

If f_i is nonconvex, but twice differentiable, then by [84, Lemma 1.2.2] the minimum eigenvalue of Hessian matrix is no smaller than $-L$, thus $L_i(x_i, \boldsymbol{z}, \boldsymbol{\lambda})$ is strongly convex with modulus $\rho - L$, given that $\rho \geq L$. Therefore, we have

$$L(\boldsymbol{x}^{k+1}) - L(\boldsymbol{x}^k) \leq \sum_{i=1}^{N} \frac{D_i - \rho D_i + L}{2} \|x_i^k - x_i^{k+1}\|^2 - \frac{\rho^2}{2} \sum_{i=1}^{N} \sum_{j \sim i} \|\hat{z}_j^k - z_j^k\|^2. \quad (5.19)$$

If f_i is strongly with modulus σ_i, by similar argument, we have

$$L(\boldsymbol{x}^{k+1}) - L(\boldsymbol{x}^k) \leq \sum_{i=1}^{N} \frac{D_i - \rho D_i - \sigma}{2} \|x_i^k - x_i^{k+1}\|^2 - \frac{\rho^2}{2} \sum_{i=1}^{N} \sum_{j \sim i} \|\hat{z}_j^k - z_j^k\|^2 \quad (5.20)$$

where $\sigma = \min_i \sigma_i$.

Similar to x update, the Lagrangian is also separable across z_j, yielding the per-master update in the form of

$$L_j(\boldsymbol{x}^{k+1}, z_j^{k+1}, \boldsymbol{\lambda}^k) = h_j(z_j^{k+1}) - v_j^k z_j^{k+1} + \frac{\rho}{2}(E_j \|z_j^{k+1}\|^2 - 2 z_j^{k+1} \sum_{i \in \mathcal{A}_j^k} x_i^{k+1}) + \frac{\gamma}{2} \|z_j^{k+1} - z_j^k\|^2 \quad (5.21)$$

Since $h_j(\cdot)$ is convex, then $L_j(\boldsymbol{x}, z_j, \boldsymbol{\lambda}) + \gamma/2 \|z_j - z_j^k\|^2$ is strongly convex with modulus $(\rho E_j + \gamma)/2$. A direct consequence of strong convexity is

$$L_j(\boldsymbol{x}^{k+1}, z_j^k, \boldsymbol{\lambda}^k) - L_j(\boldsymbol{x}^{k+1}, z_j^{k+1}, \boldsymbol{\lambda}^k) - \frac{\gamma}{2} \|z_j^{k+1} - z_j^k\|^2 \geq \partial_{z_j} L_j(\cdot)^\top (z_j^k - z_j^{k+1}) + \frac{(\rho E_j + \gamma)}{2} \|z_j^k - z_j^{k+1}\|^2$$

where $\partial_{z_j} L_j(\cdot)$ denotes the subdifferential of $L_j(\cdot)$. The optimality condition leads to $\partial_{z_j} L_j(\cdot)^\top (z_j^k - z_j^{k+1}) \geq 0$, and the definition of iteration counter implies only one master is updated during one iteration, denoted by j_k, thence we obtain

$$L(\boldsymbol{x}^{k+1}, \boldsymbol{z}^{k+1}, \boldsymbol{\lambda}^k) - L(\boldsymbol{x}^{k+1}, \boldsymbol{z}^k, \boldsymbol{\lambda}^k) \leq -\frac{\rho E_{j_k} + 2\gamma}{2} \|z_{j_k}^k - z_{j_k}^{k+1}\|^2 \quad (5.22)$$

where 2γ is due to the fact that $\frac{\gamma}{2} \|z_j^{k+1} - z_j^k\|^2$ is not included in the definition of Lagrangian function.

Since λ updates always happen after master updates, the information used is always the latest. Hence, we can bound λ by

$$\begin{aligned} & L(\boldsymbol{x}^{k+1}, \boldsymbol{z}^{k+1}, \boldsymbol{\lambda}^{k+1}) - L(\boldsymbol{x}^{k+1}, \boldsymbol{z}^{k+1}, \boldsymbol{\lambda}^{k}) \\ = & (\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^{k})^\top (\boldsymbol{A}\boldsymbol{x}^{k+1} - \boldsymbol{B}\boldsymbol{z}^{k+1}) \\ = & \sum_t (\lambda_t^{k+1} - \lambda_t^k)^\top (x_i^{k+1} - z_j^{k+1}),\ t = \mathcal{T}(i,j) \\ = & \frac{1}{\rho} \sum_t \|\lambda_t^{k+1} - \lambda_t^k\|^2,\ t = \mathcal{T}(i,j) \\ = & \frac{1}{\rho} \|\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k\|^2 \end{aligned} \tag{5.23}$$

where the fourth equality is the direct consequence of λ update, the last equality is due to the fact that only λ associated with $i \in \mathcal{A}_j^k$ and j are updated at k-th iteration.

Adding three terms (5.18)(5.22)(5.23) together yields (5.11).

5.9 Proof of Lemma 2

Lemma 1 (5.11) provides a way to bound the difference of Lagrangian value of two consecutive iterations through the difference of primal and dual variables. Among all variable differences, $\|\hat{z}_j^k - z_j^k\|^2$ is the only term coming from asynchronism of the algorithm. To effectively characterize the bound, we need to a bound that only involves two adjacent iterations. We first explicitly expand $\hat{z}_j^k$ as a delayed version of z_j^k, that is, $\hat{z}_j^k = z_j^{k-\tau_{ij}}$, where τ_{ij} denotes the value of z_j used to update $x_i^k,\ i \in \mathcal{A}_j^k$ just before the current update. Using basic inequality, we arrive at

$$\|\hat{z}_j^k - z_j^k\|^2 = \left\| \sum_{l=k-\tau}^{k-1} (z_j^l - z_j^{l+1}) \right\|^2 \leq (\tau - 1) \sum_{l=k-\tau}^{k-1} \|z_j^l - z_j^{l+1}\|^2. \tag{5.24}$$

Further assuming the cardinality of active set is below threshold for each iteration, i.e., $|\mathcal{A}_j^k| < S$, then we obtain

$$\sum_{i \in \mathcal{A}_j^k} \sum_{j \sim i} \|\hat{z}_j^k - z_j^k\|^2 \leq S(\tau - 1) \sum_{l=k-\tau}^{k-1} \sum_{j \sim i} \|z_j^l - z_j^{l+1}\|^2 \leq S(\tau - 1) \sum_{l=k-\tau}^{k-1} \|z_{j_l}^l - z_{j_l}^{l+1}\|^2.$$

Summing both sides over iteration $k = 1, 2, \ldots, K$, we arrive at

$$\sum_{k=0}^{K} \sum_{i \in \mathcal{A}_j^k} \sum_{j \sim i} \|\hat{z}_j^k - z_j^k\|^2 \leq S(\tau - 1) \sum_{k=0}^{K} \sum_{l=k-\tau}^{k-1} \|z_{j_l}^l - z_{j_l}^{l+1}\|^2. \tag{5.25}$$

Notice that in the righthand side, each term $\|z_{j_l}^l - z_{j_l}^{l+1}\|^2$ can appear at most $\tau - 1$ times. Therefore, we can further bound the summation by

$$\sum_{k=0}^{K} \sum_{l=k-\tau}^{k-1} \|z_{j_l}^l - z_{j_l}^{l+1}\|^2 \leq (\tau - 1) \sum_{k=0}^{K} \|z_{j_k}^k - z_{j_k}^{k+1}\|^2. \tag{5.26}$$

Combining (5.26) and (5.25), we arrive at (5.12).

5.10 Proof of Theorem 1

Notice that at iteration k, only master j_k is updated, and so are $\lambda_t,\ t = \mathcal{T}(i, j_k)$. We conclude that

$$\|u_i^{k+1} - u_i^k\|^2 \leq \|\nabla f_i(x_i^{k+1}) - \nabla f_i(x_i^k)\|^2 \leq L^2 \|x_i^{k+1} - x_i^k\|^2. \tag{5.27}$$

Since only one master can update during each iteration, it holds that

$$\|\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k\|^2 \leq L^2 \sum_{i \in \mathcal{A}_j^k} \|x_i^{k+1} - x_i^k\|^2 \tag{5.28}$$

where we have used the fact that

$$u_i^k - u_i^{k+1} = \sum_t (\lambda_t^k - \lambda_t^{k+1}),\ t = \mathcal{T}(i, j).$$

The goal is to bound the distance of Lagrangian value to its initial value. By summing (5.11)

over k and using Lemma 11, we have

$$
\begin{aligned}
& L(\boldsymbol{x}^{k+1}, \boldsymbol{z}^{k+1}, \boldsymbol{\lambda}^{k+1}) - L(\boldsymbol{x}^0, \boldsymbol{z}^0, \boldsymbol{\lambda}^0) \\
\leq & -\sum_{l=0}^{k} \frac{\rho E_{j_l} + 2\gamma}{2} \|z_{j_l}^l - z_{j_l}^{l+1}\|^2 + \frac{1}{\rho} \sum_{l=0}^{k} \|\boldsymbol{\lambda}^{l+1} - \boldsymbol{\lambda}^l\|^2 \\
& + \frac{\rho^2}{2} \sum_{l=0}^{k} \sum_{i \in \mathcal{A}_j^l} \sum_{j \sim i} \|\hat{z}_j^l - z_j^l\|^2 + \sum_{l=0}^{k} \sum_{i \in \mathcal{A}_j^l} \frac{D_i - \rho D_i}{2} \|x_i^{l+1} - x_i^l\|^2 \\
\leq & \sum_{l=0}^{k} \frac{\rho^2 S(\tau-1)^2 - \rho E_{j_l} - 2\gamma}{2} \|z_{j_l}^l - z_{j_l}^{l+1}\|^2 + \frac{\rho D_{\min} - \rho^2 D_{\min} + 2L^2}{2\rho} \sum_{l=0}^{k} \sum_{i \in \mathcal{A}_j^l} \|x_i^{l+1} - x_i^l\|^2.
\end{aligned}
$$

To ensure each term is positive, we need to have

$$
\begin{aligned}
S(\tau-1)^2 - \rho E_{\min} - 2\gamma &\leq 0 \\
\rho D_{\min} - \rho^2 D_{\min} + 2L^2 &\leq 0
\end{aligned}
$$

which yields

$$
\rho \geq \frac{D_{\min} + L + \sqrt{(D_{\min} + L)^2 + 8L^2 D_{\min}}}{2D_{\min}} \tag{5.29}
$$

$$
\gamma \geq \frac{S(\tau-1)^2 \rho^2 - \rho E_{\min}}{2}. \tag{5.30}
$$

Chapter 6

Summary and Future Directions

The present thesis contributes to the efficient and scalable design of ADMM-based algorithms tailored for decentralized optimization. In this final chapter, the main results are summarized and possible future directions are discussed.

6.1 Thesis summary

Chapter 2 deals with decentralized optimization problems where no global coordinator exists and every node can only talk to its immediate neighbors. Many problems in signal processing and machine learning can be casted in this form and ADMM is particularly suitable for such problems thanks to its decomposability. Conventional fully decentralized approaches handle one neighbor at a time without realizing the "big picture" of local topology information. The proposed hybrid ADMM method offers a unifying framework that subsumes both centralized and decentralized ADMM as special cases, and together with so called "in-network acceleration" it can take advantage of topology of local neighborhood. The numerical experiments on various graphs showcase the merits the proposed method.

Chapter 3 explores ways to improve the performance of hybrid ADMM by taking into consideration the weights of updates. In decentralized settings, not all neighbors are of the same importance, so it is quite natural to assign larger weights to more critical neighbors. The weights may correspond to physical quantities such as link speed or cost, but they may also be implemented algorithmically at each node, making them applicable for all scenarios where hybrid ADMM is applicable. Then the convergence rate of weighted hybrid ADMM is established and

it is shown that some parameters are closely related to the spectral properties of the underlying graph. Thus, optimizing the convergence rate with respect to the graph related parameters reveals itself better weights. The effectiveness of proposed method is validated numerically.

Chapter 4 also explores ways to handle weighted updates from a different perspective. It is shown that weighted updates are mathematically equivalent to preconditioning ADMM, whose convergence rate is dependent on the condition number of the Laplacian matrix the underlying communication graph. However, it can be quite convoluted trying to directly optimize this rate through tuning weights because the dependency of convergence rate on the Laplacian matrix is too complicated. Instead of attacking the problem directly, a surrogate obtained using Schur complement is proposed and shown to be the upper bound of original solution. Hence, the preconditioning problem can be solved by optimizing this surrogate. The numerical results reported demonstrates that preconditioning could greatly improve the convergence speed of decentralized optimization.

While all previous chapters consider problems in synchronous settings where all updates are timed perfectly, Chapter 5 studies the more practical case where there is no synchronization among nodes, which aries frequently in practical systems due to reasons such as unreliable communication, time-varying topology, prohibitive cost, or the lack of synchronized clock. Asynchronous algorithms effectively mitigate the straggler problem and offer great improvement relative to synchronous counterparts since faster machines do not have to idle to wait for slower ones. This chapter proposes asynchronous hybrid ADMM allowing the existence of multiple masters, which can not be handled by any conventional methods. With some mild assumptions, it is established that convergence of asynchronous hybrid ADMM still holds, which means the techniques developed in previous chapters are also applicable. The efficiency of asynchronous hybrid ADMM is further demonstrated through experiments.

6.2 Future directions

The contributions in this thesis open up some interesting directions to explore and open problems to be solved. In the following, some possible research directions will be briefly discussed.

- **Optimal centrality measurements.** The node importance measurement used to create virtual fusion centers in [71] is *degree centrality*, the number of neighbors a node is connected to. The rational of using this centrality measurement is that a node with large

number of neighbors acting like a central coordinator could possibly facilitate the flow of information and thus speed up the convergence. However, there is no optimality guarantee for node centrality and there are other centrality measurements out there, see e.g., [8, 111, 101], potentially with better performance. Systematic ways to design the node and edge centrality with some optimality guarantees could be one important research topic.

- **Hierarchical structured network.** The communication graph of hybrid ADMM can be modeled as hypergraphs, which can be seen as two layer networks when the hyperedges are viewed as parent nodes. However, since the problem formulation only involves constraints of the form $\boldsymbol{x}_i = \boldsymbol{z}_j$, no communication among group coordinators is allowed. As a result, it cannot handle the cases when group coordinators are directly connected. More importantly, such limitation prevent it from handling hierarchical networks that arise frequently in large-scale networks such as computer network or cellular network. Take the simple two-cluster graph for example. If the first cluster center is selected as group center, then the second cluster center cannot be chosen as group center anymore because it belongs to the neighborhood of the first cluster center. Therefore, an algorithm that naturally works for hierarchical structure would be an intersting direction to explore.

- **Convergence with practical parameter requirement.** Chapter 5 proposed the AH-ADMM algorithm and established convergence for nonconvex problem following the precursor [16]. However, the convergence proof requires adding one proximal term and setting the algorithm parameter γ to be sufficiently large, usually too large for to be practical. In fact, preliminary results show that any positive γ works well, which is not well aligned with the convergence theory, and setting the parameter values suggested by theory actually leads to worse performance. Therefore, the future work should focus on a convergence proof that is on par with its synchronous counterparts.

- **Inexact updates.** So far, all ADMM iterations are assumed to be carried out exactly. This is easy when the subproblem admits closed form solutions, but can be problematic when it does not. In such cases, an iterative subroutine could be employed to obtain an approximate solution. Inexact updates have been studied for ADMM, see [15, 17]. Future research should explore this direction and quantify the impact of errors in updates on the accuracy of final solution.

References

[1] A. Abboud, R. Couillet, M. Debbah, and H. Siguerdidjane, "Asynchronous alternating direction method of multipliers applied to the direct-current optimal power flow problem," in *Proc. of Intl. Conf. on Acoust., Speech, and Signal Processing*, May 2014, pp. 7764–7768.

[2] A. Agarwal and J. C. Duchi, "Distributed Delayed Stochastic Optimization," in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 873–881.

[3] J. A. Bazerque and G. B. Giannakis, "Distributed Spectrum Sensing for Cognitive Radio Networks by Exploiting Sparsity," *IEEE Trans. Signal Process.*, vol. 58, no. 3, pp. 1847–1862, Mar. 2010.

[4] J. A. Bazerque, G. Mateos, and G. B. Giannakis, "Group-lasso on splines for spectrum cartography," *IEEE Trans. Signal Process.*, vol. 59, no. 10, pp. 4648–4663, 2011.

[5] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Englewood Cliffs, NJ: Prentice-Hall, 1989, vol. 23.

[6] D. Boley, "Local linear convergence of the alternating direction method of multipliers on quadratic or linear programs," *SIAM J. Optim.*, vol. 23, no. 4, pp. 2183–2207, 2013.

[7] M. Bolla, "Spectra, Euclidean representations and clusterings of hypergraphs," *Discrete Mathematics*, vol. 117, no. 1, pp. 19–39, Jul. 1993.

[8] S. P. Borgatti and M. G. Everett, "A Graph-theoretic perspective on centrality," *Social Networks*, vol. 28, no. 4, pp. 466–484, Oct. 2006.

[9] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed Optimization and Statistical Learning via the Alternating Mirection Method of Multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[10] S. P. Boyd, L. E. Ghaoui, E. Feron, and V. Balakrishnan, *Linear Matrix Inequalities in System and Control Theory*, ser. SIAM Studies in Applied Mathematics. Philadelphia, PA: SIAM, 1994, no. vol. 15.

[11] R. Braatz and M. Morari, "Minimizing the Euclidean Condition Number," *SIAM J. Control Optim.*, vol. 32, no. 6, pp. 1763–1768, Nov. 1994.

[12] A. Bretto, *Hypergraph Theory: An Introduction*. Springer Publishing Company, Incorporated, 2013.

[13] Y. Cao, W. Yu, W. Ren, and G. Chen, "An overview of recent progress in the study of distributed multi-agent coordination," *IEEE Trans. Ind. Informat.*, vol. 9, no. 1, pp. 427–438, 2013.

[14] T. Chang, M. Hong, W. Liao, and X. Wang, "Asynchronous Distributed ADMM for Large-Scale Optimization—Part I: Algorithm and Convergence Analysis," *IEEE Trans. Signal Process.*, vol. 64, no. 12, pp. 3118–3130, Jun. 2016.

[15] T. H. Chang, M. Hong, and X. Wang, "Multi-Agent Distributed Optimization via Inexact Consensus ADMM," *IEEE Trans. Signal Process.*, vol. 63, no. 2, pp. 482–497, Jan. 2015.

[16] T. H. Chang, W. C. Liao, M. Hong, and X. Wang, "Asynchronous Distributed ADMM for Large-Scale Optimization—Part II: Linear Convergence Analysis and Numerical Performance," *IEEE Trans. Signal Process.*, vol. 64, no. 12, pp. 3131–3144, Jun. 2016.

[17] T.-H. Chang, M. Hong, and X. Wang, "Multi-agent distributed large-scale optimization by inexact consensus alternating direction method of multipliers," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 6137–6141.

[18] F. Chung, *Spectral Graph Theory*, ser. CBMS Regional Conference Series in Mathematics. American Mathematical Society, 1997, vol. 92.

[19] W. Deng and W. Yin, "On the Global and Linear Convergence of the Generalized Alternating Direction Method of Multipliers," *J. Sci. Comput.*, vol. 66, no. 3, pp. 889–916, May 2015.

[20] A. G. Dimakis, S. Kar, J. M. F. Moura, M. G. Rabbat, and A. Scaglione, "Gossip Algorithms for Distributed Signal Processing," *Proc. IEEE Proc.*, vol. 98, no. 11, pp. 1847–1864, Nov. 2010.

[21] J. C. Duchi, A. Agarwal, and M. J. Wainwright, "Dual Averaging for Distributed Optimization: Convergence Analysis and Network Scaling," *IEEE Trans. Autom. Control*, vol. 57, no. 3, pp. 592–606, Mar. 2012.

[22] J. Eckstein, "Splitting methods for monotone operators with applications to parallel optimization," Thesis, Massachusetts Institute of Technology, 1989.

[23] J. Eckstein and D. P. Bertsekas, "On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators," *Mathematical Programming*, vol. 55, no. 1-3, pp. 293–318, Apr. 1992.

[24] T. Erseghe, D. Zennaro, E. Dall'Anese, and L. Vangelista, "Fast Consensus by the Alternating Direction Multipliers Method," *IEEE Trans. Signal Process.*, vol. 59, no. 11, pp. 5523–5537, Nov. 2011.

[25] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On Power-law Relationships of the Internet Topology," in *Proc. of Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '99. New York, NY, USA: ACM, 1999, pp. 251–262.

[26] F. Farina, A. Garulli, A. Giannitrapani, and G. Notarstefano, "Distributed Constrained Nonconvex Optimization: The Asynchronous Method of Multipliers," *arXiv preprent:1806.05181*, Jun. 2018.

[27] P. A. Forero, A. Cano, and G. B. Giannakis, "Distributed Clustering Using Wireless Sensor Networks," *IEEE J. Sel. Topics Signal Process.*, vol. 5, no. 4, pp. 707–724, Aug. 2011.

[28] ——, "Consensus-Based Distributed Support Vector Machines," *J. Mach. Learn. Res.*, vol. 11, no. May, pp. 1663–1707, 2010.

[29] ——, "Convergence analysis of consensus-based distributed clustering." in *Proc. of Intl. Conf. on Acoust., Speech, and Signal Processing*, Dallas, TX, USA, 2010, pp. 1890–1893.

[30] G. França and J. Bento, "Distributed Optimization, Averaging via ADMM, and Network Topology," *Proc. IEEE*, vol. 108, no. 11, pp. 1939–1952, Nov. 2020.

[31] ——, "How is Distributed ADMM Affected by Network Topology?" *arXiv preprint:1710.00889*, Oct. 2017.

[32] D. Gabay, "Applications of the Method of Multipliers to Variational Inequalities," in *Studies in Mathematics and Its Applications*, ser. Augmented Lagrangian Methods: Applications to the Numerical Solution of Boundary-Value Problems, M. Fortin and R. Glowinski, Eds. Amsterdam: North-Holland, 1983, vol. 15, pp. 299–331.

[33] D. Gabay and B. Mercier, "A dual algorithm for the solution of nonlinear variational problems via finite element approximation," *Computers & Mathematics with Applications*, vol. 2, no. 1, pp. 17–40, Jan. 1976.

[34] L. Gan, U. Topcu, and S. H. Low, "Optimal decentralized protocol for electric vehicle charging," *IEEE Trans. Power Syst.*, vol. 28, no. 2, pp. 940–951, May 2013.

[35] E. Ghadimi, A. Teixeira, M. G. Rabbat, and M. Johansson, "The ADMM algorithm for distributed averaging: Convergence rates and optimal parameter selection," in *Proc. of Asilomar Conf. on Signals, Systems, and Computers*, Nov. 2014, pp. 783–787.

[36] B. Gharesifard and J. Cortés, "Distributed Continuous-Time Convex Optimization on Weight-Balanced Digraphs," *IEEE Trans. Autom. Control*, vol. 59, no. 3, pp. 781–786, Mar. 2014.

[37] G. B. Giannakis, Q. Ling, G. Mateos, I. D. Schizas, and H. Zhu, "Decentralized Learning for Wireless Communications and Networking," in *Splitting Methods in Communication, Imaging, Science, and Engineering*, R. Glowinski, S. J. Osher, and W. Yin, Eds. Cham: Springer, 2016, pp. 461–497.

[38] P. Giselsson, "Tight global linear convergence rate bounds for Douglas–Rachford splitting," *Journal of Fixed Point Theory and Applications*, vol. 19, no. 4, pp. 2241–2270, Dec. 2017.

[39] P. Giselsson and S. Boyd, "Diagonal scaling in Douglas-Rachford splitting and ADMM," in *Proc. of Conf. on Decision and Control*, Los Angeles, CA, USA, Dec. 2014, pp. 5033–5039.

[40] ——, "Metric selection in fast dual forward–backward splitting," *Automatica*, vol. 62, pp. 1–10, Dec. 2015.

[41] ——, "Linear Convergence and Metric Selection for Douglas-Rachford Splitting and ADMM," *IEEE Trans. Autom. Control*, vol. 62, no. 2, pp. 532–544, Feburary 2017.

[42] R. Glowinski and A. Marroco, "Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de Dirichlet non linéaires," *Revue française d'automatique, informatique, recherche opérationnelle. Analyse numérique*, vol. 9, no. R2, pp. 41–76, 1975.

[43] T. Goldstein, B. O'Donoghue, S. Setzer, and R. Baraniuk, "Fast alternating direction optimization methods," *SIAM J. Imaging Sci.*, vol. 7, no. 3, pp. 1588–1623, 2014.

[44] M. Grant and S. Boyd, "CVX: Matlab Software for Disciplined Convex Programming, version 2.1," Mar. 2014.

[45] M. C. Grant and S. P. Boyd, "Graph Implementations for Nonsmooth Convex Programs," in *Recent Advances in Learning and Control*, ser. Lecture Notes in Control and Information Sciences, V. D. Blondel, S. P. Boyd, and H. Kimura, Eds. London: Springer, 2008, pp. 95–110.

[46] J. P. Gross, R. B. Pandit, C. G. Cook, and T. H. Matson, "Network with distributed shared memory," US Patent US9 317 469B2, Apr., 2016.

[47] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013.

[48] R. Hannah and W. Yin, "On Unbounded Delays in Asynchronous Parallel Fixed-Point Algorithms," *J. Sci. Comput.*, pp. 1–28, Dec. 2017.

[49] M. Hong, "A Distributed, Asynchronous and Incremental Algorithm for Nonconvex Optimization: An ADMM Approach," *IEEE Control Netw. Syst.*, 2017.

[50] M. Hong, Z. Luo, and M. Razaviyayn, "Convergence Analysis of Alternating Direction Method of Multipliers for a Family of Nonconvex Problems," *SIAM J. Optim.*, vol. 26, no. 1, pp. 337–364, Jan. 2016.

[51] M. Hong and Z. Q. Luo, "On the linear convergence of the alternating direction method of multipliers," *Math. Program.*, vol. 162, no. 1-2, pp. 165–199, Mar. 2017.

[52] S. Hosseini, A. Chapman, and M. Mesbahi, "Online distributed ADMM via dual averaging," in *Proc. of Conf. on Decision and Control*, Dec. 2014, pp. 904–909.

[53] F. Iutzeler, P. Bianchi, P. Ciblat, and W. Hachem, "Asynchronous distributed optimization using a randomized alternating direction method of multipliers," in *Proc. of Conf. on Decision and Control*, Florence, Italy, Dec. 2013, pp. 3671–3676.

[54] ——, "Explicit Convergence Rate of a Distributed Alternating Direction Method of Multipliers," *IEEE Trans. Autom. Control*, vol. 61, no. 4, pp. 892–904, Apr. 2016.

[55] A. K. Jain, *Fundamentals of Digital Image Processing*. USA: Prentice-Hall, Inc., 1989.

[56] D. Jakovetic, J. Xavier, and J. M. F. Moura, "Cooperative Convex Optimization in Networked Systems: Augmented Lagrangian Algorithms With Directed Gossip Communication," *IEEE Trans. Signal Process.*, vol. 59, no. 8, pp. 3889–3902, Aug. 2011.

[57] V. Kekatos and G. B. Giannakis, "Distributed robust power system state estimation," *IEEE Trans. Power Syst*, vol. 28, no. 2, pp. 1617–1626, 2013.

[58] E. Kokiopoulou and P. Frossard, "Distributed Classification of Multiple Observation Sets by Consensus," *IEEE Trans. Signal Process.*, vol. 59, no. 1, pp. 104–114, Jan. 2011.

[59] H. Kopetz and W. Ochsenreiter, "Clock Synchronization in Distributed Real-Time Systems," *IEEE Trans. Comput.*, vol. C-36, no. 8, pp. 933–940, Aug. 1987.

[60] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B. Y. Su, "Scaling Distributed Machine Learning with the Parameter Server," in *Proc. of the USENIX Conf. on Operating Sys. Design and Implementation*, vol. 14, 2014, pp. 583–598.

[61] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous Parallel Stochastic Gradient for Nonconvex Optimization," in *Proc. of Neural Information Processing (NeurIPS)*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, pp. 2737–2745.

[62] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can Decentralized Algorithms Outperform Centralized Algorithms? A Case Study for Decentralized Parallel Stochastic Gradient Descent," in *Proc. of Neural Information Processing (NeurIPS)*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5330–5340.

[63] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous Decentralized Parallel Stochastic Gradient Descent," in *Proc. of Intl. Conf. on Machine Learning*. PMLR, Jul. 2018, pp. 3043–3052.

[64] T. Lin, S. Ma, and S. Zhang, "On the Global Linear Convergence of the ADMM with MultiBlock Variables," *SIAM J. Optim.*, vol. 25, no. 3, pp. 1478–1497, Jan. 2015.

[65] Q. Ling, Y. Liu, W. Shi, and Z. Tian, "Communication-efficient weighted ADMM for decentralized network optimization," in *Proc. of Intl. Conf. on Acoust., Speech, and Signal Processing*. Shanghai, China: IEEE, 2016, pp. 4821–4825.

[66] Q. Ling and Z. Tian, "Decentralized Sparse Signal Recovery for Compressive Sleeping Wireless Sensor Networks," *IEEE Trans. Signal Process.*, vol. 58, no. 7, pp. 3816–3827, Jul. 2010.

[67] P. Lions and B. Mercier, "Splitting Algorithms for the Sum of Two Nonlinear Operators," *SIAM J. Numer. Anal.*, vol. 16, no. 6, pp. 964–979, Dec. 1979.

[68] J. S. Liu and C. H. R. Lin, "Asynchronous Distributed Joint Optimization in Wireless Multi-Hop Networks," *IEEE Communications Letters*, vol. 19, no. 9, pp. 1620–1623, Sep. 2015.

[69] M. Ma and G. B. Giannakis, "Graph-aware Weighted Hybrid ADMM for Fast Decentralized Optimization," in *Proc. of Asilomar Conf. on Signals, Systems, and Computers*, Oct. 2018, pp. 1881–1885.

[70] ——, "Preconditioning ADMM for Fast Decentralized Optimization," in *Proc. of Intl. Conf. on Acoust., Speech, and Signal Processing*, May 2020, pp. 3142–3146.

[71] M. Ma, A. N. Nikolakopoulos, and G. B. Giannakis, "Hybrid ADMM: A unifying and fast approach to decentralized optimization," *EURASIP J. Adv. Signal Process.*, vol. 2018, no. 1, p. 73, 2018.

[72] M. Ma, J. Ren, G. B. Giannakis, and J. Haup, "Fast Asynchronous Decentralized Optimization: Allowing Multiple Masters," in *Proc. of Global Conf. on Signal and Informatino Processing*, Nov. 2018, pp. 633–637.

[73] M. Ma, A. N. Nikolakopoulos, and G. B. Giannakis, "Fast Decentralized Learning via Hybrid Consensus ADMM," in *Proc. of Intl. Conf. on Acoust., Speech, and Signal Processing*, Calgary, Alberta, Canada, Apr. 2018.

[74] A. Makhdoumi and A. Ozdaglar, "Convergence Rate of Distributed ADMM Over Networks," *IEEE Trans. Autom. Control*, vol. 62, no. 10, pp. 5082–5095, Oct. 2017.

[75] S. K. Mani, R. Durairajan, P. Barford, and J. Sommers, "A System for Clock Synchronization in an Internet of Things," *arXiv preprint:1806.02474*, Jun. 2018.

[76] X. Mao, K. Yuan, Y. Hu, Y. Gu, A. H. Sayed, and W. Yin, "Walkman: A Communication-Efficient Random-Walk Algorithm for Decentralized Optimization," *IEEE Trans. Signal Process.*, vol. 68, pp. 2513–2528, 2020.

[77] G. Mateos, J. A. Bazerque, and G. B. Giannakis, "Distributed Sparse Linear Regression," *IEEE Trans. Signal Process.*, vol. 58, no. 10, pp. 5262–5276, Oct. 2010.

[78] G. Mateos, I. D. Schizas, and G. B. Giannakis, "Distributed Recursive Least-Squares for Consensus-Based In-Network Adaptive Estimation," *IEEE Trans. Signal Process.*, vol. 57, no. 11, pp. 4583–4588, Nov. 2009.

[79] J. F. C. Mota, J. M. F. Xavier, P. M. Q. Aguiar, and M. Püschel, "D-ADMM: A Communication-Efficient Distributed Algorithm for Separable Optimization," *IEEE Trans. Signal Process.*, vol. 61, no. 10, pp. 2718–2723, May 2013.

[80] L. Narula and T. E. Humphreys, "Requirements for Secure Clock Synchronization," *IEEE J. Sel. Topics Signal Process.*, vol. 12, no. 4, pp. 749–762, Aug. 2018.

[81] A. Nedić, D. P. Bertsekas, and V. S. Borkar, "Distributed asynchronous incremental subgradient methods," *Studies in Computational Mathematics*, vol. 8, no. C, pp. 381–407, 2001.

[82] A. Nedic and A. Ozdaglar, "Distributed Subgradient Methods for Multi-Agent Optimization," *IEEE Trans. Autom. Control*, vol. 54, no. 1, pp. 48–61, Jan. 2009.

[83] A. Nedic, A. Ozdaglar, and P. A. Parrilo, "Constrained consensus and optimization in multi-agent networks," *IEEE Trans. Autom. Control*, vol. 55, no. 4, pp. 922–938, 2010.

[84] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*, ser. Applied Optimization. Springer US, 2004.

[85] M. Newman, "The Structure and Function of Complex Networks," *SIAM Rev.*, vol. 45, no. 2, pp. 167–256, Jan. 2003.

[86] A. N. Nikolakopoulos and J. D. Garofalakis, "On the Multi-Level Near Complete Decomposability of a Class of Multiprocessing Systems," in *Proceedings of the Pan-Hellenic Conference on Informatics*, ser. PCI '16. New York, NY, USA: ACM, 2016, pp. 14:1–14:6.

[87] R. Nishihara, L. Lessard, B. Recht, A. Packard, and M. Jordan, "A General Analysis of the Convergence of ADMM," in *Proc. of Intl. Conf. on Machine Learning*. PMLR, Jun. 2015, pp. 343–352.

[88] I. Notarnicola and G. Notarstefano, "Asynchronous Distributed Optimization Via Randomized Dual Proximal Gradient," *IEEE Trans. Autom. Control*, vol. 62, no. 5, pp. 2095–2106, May 2017.

[89] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and Cooperation in Networked Multi-Agent Systems," *Proc. IEEE Proc.*, vol. 95, no. 1, pp. 215–233, Jan. 2007.

[90] R. Olfati-Saber and J. S. Shamma, "Consensus Filters for Sensor Networks and Distributed Sensor Fusion," in *Proc. of Conf. on Decision and Control*. Seville, Spain: IEEE, Dec. 2005, pp. 6698–6703.

[91] Z. Peng, T. Wu, Y. Xu, M. Yan, and W. Yin, "Coordinate Friendly Structures, Algorithms and Applications," *Annals of Mathematical Sciences and Applications*, vol. 1, no. 1, pp. 57–119, 2016.

[92] J. B. Predd, S. R. Kulkarni, and H. V. Poor, "A Collaborative Training Algorithm for Distributed Learning," *IEEE Trans. Inf. Theory*, vol. 55, no. 4, pp. 1856–1871, Apr. 2009.

[93] M. Rabbat and R. Nowak, "Distributed optimization in sensor networks," in *Proc. of Intl. Conf. on Information Processing in Sensor Networks*. Berkeley, California, USA: ACM Press, 2004, p. 20.

[94] S. S. Ram, A. Nedić, and V. V. Veeravalli, "Asynchronous gossip algorithms for stochastic optimization," in *Proc. of Conf. on Decision and Control*, Dec. 2009, pp. 3581–3586.

[95] C. Ravazzi, S. M. Fosson, and E. Magli, "Distributed soft thresholding for sparse signal recovery," in *2013 IEEE Global Communications Conference (GLOBECOM)*, 2013, pp. 3429–3434.

[96] J. Ren, X. Li, and J. Haupt, "Communication-Efficient distributed optimization for sparse learning via two-way truncation," in *Proc. of CAMSAP*, Curacao, Dec. 2017, pp. 1–5.

[97] W. Ren, "Consensus based formation control strategies for multi-vehicle systems," in *2006 American Control Conference*, Minneapolis, MN, USA, Jun. 2006, pp. 4237–4242.

[98] W. Ren, R. W. Beard, and E. M. Atkins, "Information consensus in multivehicle cooperative control," *IEEE Control Syst. Mag.*, vol. 27, no. 2, pp. 71–82, Apr. 2007.

[99] A. H. Sayed, "Adaptation, Learning, and Optimization over Networks," *Foundations and Trends® in Machine Learning*, vol. 7, no. 4-5, pp. 311–801, Jul. 2014.

[100] I. D. Schizas, A. Ribeiro, and G. B. Giannakis, "Consensus in Ad Hoc WSNs With Noisy Links – Part I: Distributed Estimation of Deterministic Signals," *IEEE Trans. Signal Process.*, vol. 56, no. 1, pp. 350–364, Jan. 2008.

[101] S. Segarra and A. Ribeiro, "Stability and Continuity of Centrality Measures in Weighted Graphs," *IEEE Trans. Signal Process.*, vol. 64, no. 3, pp. 543–555, Feb. 2016.

[102] W. Shi, Q. Ling, G. Wu, and W. Yin, "A Proximal Gradient Algorithm for Decentralized Composite Optimization," *IEEE Trans. Signal Process.*, vol. 63, no. 22, pp. 6013–6023, Nov. 2015.

[103] W. Shi, Q. Ling, K. Yuan, G. Wu, and W. Yin, "On the Linear Convergence of the ADMM in Decentralized Consensus Optimization," *IEEE Trans. Signal Process.*, vol. 62, no. 7, pp. 1750–1761, Apr. 2014.

[104] V. Smith, S. Forte, C. Ma, M. Takáč, M. I. Jordan, and M. Jaggi, "CoCoA: A General Framework for Communication-Efficient Distributed Optimization," *J. Mach. Learn. Res.*, vol. 18, no. 230, pp. 1–49, 2018.

[105] S. Sundhar Ram, A. Nedić, and V. V. Veeravalli, "Distributed Stochastic Subgradient Projection Algorithms for Convex Optimization," *J. Optim. Theory Appl.*, vol. 147, no. 3, pp. 516–545, Dec. 2010.

[106] P. Tan, "Linear Convergence Rates for Variants of the Alternating Direction Method of Multipliers in Smooth Cases," *J. Optim. Theory Appl.*, pp. 1–22, Dec. 2017.

[107] H. Terelius, U. Topcu, and R. M. Murray, "Decentralized multi-agent optimization via dual decomposition," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 11 245–11 251, 2011.

[108] K. I. Tsianos, S. Lawlor, and M. G. Rabbat, "Push-Sum Distributed Dual Averaging for convex optimization," in *Proc. of Conf. on Decision and Control*. Maui, HI, USA: IEEE, Dec. 2012, pp. 5453–5458.

[109] K. I. Tsianos and M. G. Rabbat, "Distributed strongly convex optimization," in *Proc. of Annual Allerton Conf. on Communication, Control, and Computing (Allerton)*. Monticello, IL, USA: IEEE, Oct. 2012, pp. 593–600.

[110] J. N. Tsitsiklis, "Problems in Decentralized Decision making and Computation," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, USA, Dec. 1984.

[111] T. W. Valente, K. Coronges, C. Lakon, and E. Costenbader, "How Correlated Are Network Centrality Measures?" *Connect (Tor)*, vol. 28, no. 1, pp. 16–26, Jan. 2008.

[112] R. R. S. van Lon and T. Holvoet, "When do agents outperform centralized algorithms?" *Auton. Agent Multi-Agent Syst.*, vol. 31, no. 6, pp. 1578–1609, Nov. 2017.

[113] J. G. VanAntwerp and R. D. Braatz, "A tutorial on linear and bilinear matrix inequalities," *J. Process Control*, vol. 10, no. 4, pp. 363–385, Aug. 2000.

[114] E. Wei and A. Ozdaglar, "On the O(1/k) convergence of asynchronous distributed alternating Direction Method of Multipliers," in *Proc. of Global Conf. on Signal and Info. Processing*, Austin, TX, USA, Dec. 2013, pp. 551–554.

[115] T. Wu, K. Yuan, Q. Ling, W. Yin, and A. H. Sayed, "Decentralized Consensus Optimization with Asynchrony and Delays," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 4, no. 2, pp. 293–307, 2018.

[116] L. Xiao, S. Boyd, and S. Lall, "A Scheme for Robust Distributed Sensor Fusion Based on Average Consensus," in *Proc. of Intl. Conf. on Information Processing in Sensor Networks*, ser. IPSN '05, Los Angeles, California, USA, Apr. 2005, pp. 63–70.

[117] J. Xu, S. Zhu, Y. C. Soh, and L. Xie, "A Bregman Splitting Scheme for Distributed Optimization over Networks," *IEEE Trans. Autom. Control*, vol. PP, no. 99, pp. 1–1, 2018.

[118] Z. Xu, G. Taylor, H. Li, M. A. T. Figueiredo, X. Yuan, and T. Goldstein, "Adaptive Consensus ADMM for Distributed Optimization," in *Proc. of Intl. Conf. on Machine Learning*, Jul. 2017, pp. 3841–3850.

[119] F. Zhang, Ed., *The Schur Complement and Its Applications*, ser. Numerical Methods and Algorithms. Boston, MA: Springer Science & Business Media, 2005, vol. 4.

[120] R. Zhang and J. Kwok, "Asynchronous distributed ADMM for consensus optimization," in *Proc. of Intl. Conf. on Machine Learning*, Beijing, China, Jun. 2014, pp. 1701–1709.

[121] H. Zhu, A. Cano, and G. Giannakis, "Distributed consensus-based demodulation: Algorithms and error analysis," *IEEE Trans. Wireless Commun.*, vol. 9, no. 6, pp. 2044–2054, Jun. 2010.